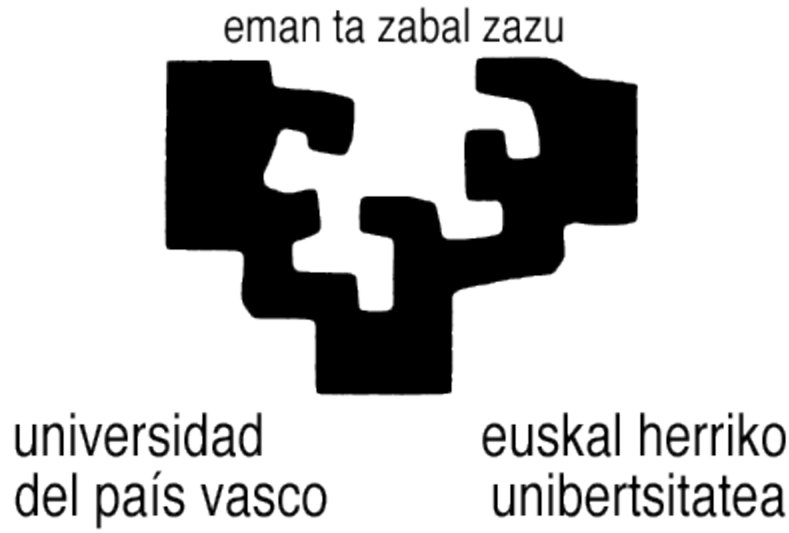

Facultad de Informática Informatika Fakultatea

TITULACIÓN: Ingeniería Técnica en Informática de Sistemas

Estudio y desarrollo de un asistente (wizard) para la generación de interfaces de usuario abstractas en entornos ubicuos.

Alumno/a: D./Dña. Francisco Javier Campos Lazkoz

Director/a: D./Dña. Julio Abascal González

Proyecto Fin de Carrera, junio de 2012

AGRADECIMIENTOS

Quiero agradecer a Raúl Miñón todo el apoyo, ayuda y disposición que me ha ofrecido durante el desarrollo de este proyecto, ha sido un lujo trabajar contigo, suerte en el futuro.

Gracias también a Julio Abascal por permitirme participar y aportar mi granito de arena en el trabajo que desarrolla el Laboratorio de Interacción Persona-Computador para Necesidades Especiales.

Lortu dugu Bruji, zu gabe ezinezkoa izango litzatzeke, mila esker.

A los familiares y amigos que han estado ahí, eskerrik asko bihotzez.

RESUMEN

En el presente proyecto se pretende desarrollar una herramienta, que tiene como objetivo asistir a diseñadores de servicios ubicuos en el proceso de generación de interfaces abstractas de usuario para entornos ubicuos. Esta herramienta facilita la tarea a los diseñadores eliminando la necesidad de que conozcan la sintaxis UIML de descripción de interfaces de usuario, ya que el documento que guarda dicha descripción se genera automáticamente tras la utilización del asistente. Este asistente permite también, enriquecer dichas descripciones ofreciendo la posibilidad de asociar a los diferentes objetos abstractos de interacción que componen la interfaz, distintos tipos de recursos multimedia. El documento UIML generado se integra en el sistema Egoki, proporcionándole la entrada necesaria para generar interfaces de usuario accesibles y adaptadas a las necesidades de las personas que desean utilizar los servicios ubicuos.

PALABRAS CLAVE: Interacción Persona Computador (IPC), Accesibilidad, Modelos de Interfaz de Usuario, Herramienta de Apoyo, Servicios Ubicuos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1 INTRODUCCIÓN

Un servicio ubicuo significa tener un servicio disponible desde cualquier lugar, en cualquier momento y de forma ininterrumpida.

Las tecnologías ubicuas permiten que dichos servicios y aplicaciones residentes en dispositivos móviles o fijos se ejecuten por si mismos de acuerdo con el contexto del usuario, y que dichos dispositivos tengan la capacidad y la inteligencia de regular el procesamiento y el intercambio de información según determinadas circunstancias.

Según esta definición se pueden considerar como ubicuos, servicios ofrecidos localmente a través de un middleware y otro tipo de servicios tales como servicios Web u otros tipos de servicios disponibles a través de Internet (Google Services, Yahoo Services,…).

Cuando en un entorno ubicuo se ofrecen diferentes servicios a través de targets, por lo general, cada uno tiene sus propios protocolos de comunicación, por lo que se requiere un middleware que sea capaz de ofrecer un protocolo de comunicación común a todos ellos y poder acceder de manera estándar a los diferentes servicios.

Además de la estandarización de los diferentes protocolos, generalmente, el middleware se encarga de las tareas de descubrimiento, comunicación y control de los diferentes servicios. Estos middleware ofrecen una interfaz abstracta describiendo las funcionalidades ofrecidas por cada uno de los servicios y en alguno de ellos, generan una interfaz de usuario simple para controlar el servicio.

Desafortunadamente, en el mejor de los casos en que el entorno ubicuo ofrezca una interfaz de usuario, esta interfaz no siempre es accesible para cualquier persona ya que no todas las personas interactúan de la misma forma y por ello cada persona tiene sus necesidades especiales.

Para evitar esta barrera en los entornos ubicuos se ha creado el sistema Egoki, el cual, a partir de la descripción abstracta que ofrece el middleware es capaz de generar, automáticamente, una interfaz de usuario final accesible y adaptada a las necesidades de las personas que desean utilizar el servicio.

Una de las características principales del sistema Egoki es que las interfaces de usuario son generadas partiendo de un lenguaje de descripción de interfaces de usuario (UIDL) [5], en concreto User Interface Modelling Language (UIML). Actualmente estas descripciones UIML se crean de forma manual. Para escribir textualmente el código UIML [1] [4] se parte de una descripción abstracta de las funcionalidades de la interfaz ofrecida por el middleware. El archivo UIML no se puede generar automáticamente a partir de la descripción abstracta, porque no se dispone de la suficiente información y, es por ello, que se debe enriquecer por el proveedor del servicio con los recursos de interacción asociados a los elementos de la interfaz abstracta y, también, ofrecer una estructura para la generación de la interfaz final.

Este proceso implica que el proveedor del servicio ubicuo tenga que crear un archivo XML acorde con la sintaxis UIML y las funcionalidades del servicio. Esta tarea, además de tediosa, no es eficaz ya que se tarda un tiempo relativamente alto para escribir

manual y textualmente un documento UIML, además la persona encargada de proveer el servicio no tiene porque estar familiarizado con la sintaxis XML, lo que podría hacer el proceso más lento.

Para solucionar este problema se propone la herramienta gráfica ASINABSUS (ASistente de INterfaces ABStractas de USuario). Esta herramienta gráfica tiene el objetivo de asistir al proveedor del servicio en el proceso de crear descripciones abstractas de interfaces de usuario, de una forma mucho más rápida e intuitiva. Además, esta herramienta permite asociar a los diferentes objetos abstractos de interacción de las interfaces, distintos tipos de recursos multimedia (texto, audio, video, imagen, etc.) permitiendo a Egoki generar interfaces basadas en las modalidades del usuario.

En el siguiente capítulo se describe la planificación realizada para desarrollar el proyecto; posteriormente se exponen en el capítulo 3 los trabajos relacionados que justifican la realización de este proyecto; en el capítulo 4 se describe el desarrollo técnico del proyecto; las pruebas diseñadas y su resultados se exponen en el capítulo 5; en el capítulo 6 se aborda la integración de la herramienta desarrollada en Egoki; finalmente, el capítulo 7 muestra las conclusiones y el trabajo futuro; tras estos apartados se ofrece la bibliografía, un glosario y un manual de usuario para guiar al diseñador en la correcta utilización de la herramienta.

CAPÍTULO 2 DESCRIPCIÓN DE OBJETIVOS DEL PROYECTO

En este capítulo se ofrece una descripción del proyecto; posteriormente se presentan los objetivos, el método de trabajo utilizado y una planificación temporal; finalmente se explica como se ha llevado a cabo la gestión.

2.1 Descripción del proyecto

En este proyecto se aborda la creación de una herramienta gráfica para asistir a proveedores de servicios ubicuos en el proceso de crear descripciones Abstractas de Interfaces de Usuario (AUI). Esta herramienta, llamada ASINABSUS, ofrecerá al proveedor una manera de crear las AUI más rápida e intuitiva que creándolas manualmente. Además evitará cometer errores en la sintaxis a la hora de diseñar las AUI y se facilitará su modificación. Después de crear la AUI, la herramienta permitirá asociar a cada Objeto Abstracto de Interacción (AIO) descrito diferentes tipos de recursos multimedia (texto, imagen, audio, video, etc.).

Finalmente la herramienta gráfica generará un documento UIML donde se proporcionará la descripción abstracta de la interfaz de usuario creada por el diseñador del servicio y los recursos asociados a los AIO que componen la interfaz.

Posteriormente, el documento UIML creado se desplegará en el sistema Egoki para, a partir de él, crear Interfaces Finales de Usuario accesibles y adaptadas a las capacidades de las personas que deseen utilizar el servicio.

2.2 Objetivos del proyecto

El objetivo principal de este proyecto es facilitar a los proveedores de servicios la provisión de documentos UIML al sistema Egoki de una manera gráfica, intuitiva y sin requerir altos conocimientos técnicos. Así mismo, para cumplir este objetivo se han identificado otros sub-objetivos:

1. Ofrecer un editor gráfico para crear AUIs con la sintaxis UIML.
2. Permitir asociar recursos multimedia a los AIO creados.
3. Generar un documento UIML que recoja la información resultante del editor y de la asociación de recursos.

Por otra parte, dado que el presente proyecto se lleva a cabo como proyecto de fin de carrera, se contemplan otros objetivos adicionales de cara al proyectando:

1. Enriquecer el conocimiento de éste a la hora de gestionar y desarrollar el ciclo de vida de un proyecto, mediante un proyecto práctico.
2. Finalizar la Diplomatura en Ingeniería técnica.
3. Estudiar nuevas tecnologías con el fin de entrar mejor preparado al mercado laboral.

2.3 Alcance

El alcance inicial del proyecto, donde se delimita el trabajo que debe realizarse para una correcta entrega de la herramienta, se describe en los siguientes puntos:

1. Parsear y transformar el documento XML donde se especifican las características del servicio ubicuo que será la entrada de información a la herramienta gráfica que se desarrollará.
2. Implementar una herramienta gráfica que asista a proveedores de servicios ubicuos en el proceso de crear una descripción abstracta de interfaz de usuario.
3. Generar un documento UIML compatible con el sistema Egoki, donde se refleje la interfaz abstracta de usuario descrita por el diseñador del servicio y los recursos asociados a sus componentes.
4. Crear una guía de instalación de la herramienta gráfica y un manual de usuario.
5. Escribir la memoria del proyecto y defenderla.

2.4 Método de trabajo

Se ha abordado un método de trabajo de un proyecto de Ingeniería dividiendo el proyecto en distintas fases: las fases que se han contemplado son la fase de gestión donde se manejan los objetivos, planificación y seguimiento del desarrollo del proyecto; la fase de formación, donde se observan las distintas herramientas y lenguajes que el proyectando debe conocer; la fase de análisis y diseño donde se contemplan las necesidades de la herramienta gráfica y como lograr que dicha herramienta cumpla con los objetivos previstos; la fase de implementación y pruebas de la herramienta gráfica, llamada fase "Aplicación"; y finalmente, la fase de cierre que trata sobre la creación de la memoria del proyecto y los manuales de usuario de la herramienta gráfica. A continuación se enumeran las tareas relacionadas con las fases mencionadas.

1. Gestión
   1.1 Objetivos: Relación de los objetivos a cumplir en este proyecto.
   1.2 Planificación: Lo relacionado con este documento.
   1.3 Seguimiento: Control del desarrollo del proyecto, posible replanificación.
2. Formación
   2.1 XML: eXtensible Markup Language, metalenguaje extensible de etiquetas que permite definir la gramática de lenguajes específicos.
   2.2 XSL: eXtensible Stylesheet Language, es una familia de lenguajes que permiten describir como los archivos codificados en XML serán formateados (para mostrarlos) o transformados.
   2.3 UIML: User Interface Markup Language, lenguaje de descripción de interfaces de usuario.
   2.4 EMF: Eclipse Modelling Framework, marco de modelado para construir herramientas basadas en un modelo de datos estructurado.
   2.5 GMF: Graphical Modelling Framework, enfoque basado en modelos para la generación de editores gráficos en Eclipse.

2.6 RCP: Rich Client Plattform, construir aplicaciones sobre la base de Eclipse.

3. Análisis y diseño
   3.1 Análisis de las necesidades de la herramienta ASINABSUS.
   3.2 Diseño del editor gráfico siguiendo la metodología del desarrollo de software dirigido por modelos.
   3.3 Diseño del módulo para asociar recursos.
   3.4 Diseño de las pruebas de funcionamiento.
4. Aplicación
   4.1 Desarrollo del editor gráfico mediante la herramienta Eclipse GMF.
   4.2 Pruebas de funcionamiento e integración del editor gráfico.
   4.3 Desarrollo del módulo de asociación mediante la herramienta Eclipse RCP.
   4.4 Pruebas funcionales.
   4.5 Integración de la aplicación con el sistema Egoki.
5. Cierre
   5.1 Creación de una memoria.
   5.2 Elaboración de manuales de uso.

2.5 Gestión y Planificación

2.5.1 Recursos

Se ha realizado una estimación de los recursos necesarios para desarrollar este proyecto, diferenciando entre recursos materiales y recursos humanos.

Recursos materiales

1. Línea ADSL 10Mb para documentación y consultas técnicas necesarias en el desarrollo del proyecto.
2. Ordenador AMD Turion con 1 GB de memoria RAM y 120 GB de disco duro. En este ordenador se ha instalado el siguiente software para el desarrollo del proyecto:
   2.1 Eclipse 3.6 Helios, entorno de desarrollo integrado de código abierto multiplataforma. Es la herramienta principal utilizada en la implementación del proyecto.
   2.2 EMF [15], framework de modelado y que facilita la generación de código para construir herramientas y otras aplicaciones basadas en un modelo de datos estructurado. Este software se ha utilizado para la creación del metamodelo de un documento UIML.
   2.3 GMF [3], proyecto Eclipse de modelado gráfico, proporciona un conjunto de componentes generadores y de las infraestructuras en tiempo de ejecución para el desarrollo de editores gráficos. Software utilizado para la creación del editor gráfico donde se describe la interfaz abstracta de usuario.
   2.4 RCP [9] [16], subconjunto de la plataforma Eclipse que consta del mínimo número de plug-ins necesarios para construir aplicaciones que no sean IDE. Utilizado en la implementación para el modulo de asociación de recursos.

2.5 Dropbox, servicio multiplataforma en la nube, operado por la compañía Dropbox. El servicio permite a los usuarios almacenar y sincronizar archivos en línea y entre computadoras y compartir archivos y carpetas con otro. Se utiliza como alojamiento de seguridad y sincronización para las partes que componen este proyecto.
2.6 Dia, editor de diagramas utilizado para la creación de los diagramas que figuran en la memoria del proyecto.
2.7 Microsoft Office Word 2003, editor de texto utilizado para redactar la presente memoria.

Recursos humanos

En la siguiente relación se muestran los recursos humanos identificados para el desarrollo del proyecto:

1. Desarrollador del proyecto, alumno de Ingeniería Técnica en Informática de Sistemas que cumpliendo los requisitos y normativa de la facultad para la finalización de la carrera, desarrolla el presente proyecto. Su función será la de elaborar el proyecto en todas sus fases: captura de requisitos, análisis, diseño, implementación, pruebas, integración y documentación. Para conseguir estos objetivos deberá adquirir los conocimientos necesarios previa documentación y estudio, siendo necesaria su actualización en todas las fases del proyecto.
2. Tutor del proyecto, persona perteneciente a la facultad del desarrollador, que coordina al mismo durante el desarrollo del proyecto haciendo uso de sus conocimientos y experiencia para que el proyecto finalice de forma satisfactoria.

2.5.2 Entregables

Aplicación ASINABSUS, mediante la cuál se entrega al diseñador del servicio ubicuo la interfaz gráfica del producto organizada en un editor que le permita crear descripciones abstractas de interfaces y una interfaz gráfica que permita al diseñador asociar a los componentes de la interfaz abstracta diseñada sus correspondientes recursos. Tras realizar estas acciones la aplicación genera un documento UIML donde se refleja la interfaz abstracta y los recursos asociados a sus componentes.

Memoria, donde se muestra la información derivada del proyecto y el ciclo de vida de su desarrollo. Se refiere a este documento.

Guías de usuario (anexo B del presente documento), donde se orienta al diseñador del servicio ubicuo en el uso de la aplicación desarrollada. Se trata de un tutorial de ayuda en el uso del asistente.

2.5.3 Fases y Tareas

Los puntos descritos en el alcance se desarrollarán en orden secuencial y dependiendo de las características de cada uno, se le asignará un ciclo de vida diferente para adaptarlo a las particulares de cada uno de ellos. También se prevé con la implementación de cada entregable, la modificación de los entregables desarrollados con anterioridad para mejorar el conjunto del proyecto y evitar posibles fallos

descubiertos durante el desarrollo del proyecto o posibles carencias anteriormente no detectadas.

Ciclo de vida de la herramienta gráfica

- Diseño de la herramienta gráfica.
- Implementación.
- Pruebas para buscar posibles errores y mejoras.
- Integración en Egoki.

Ciclo de vida de la asistencia a usuarios

- Diseño de la guía de instalación y del manual de usuario.
- Desarrollo de la guía de instalación.
- Desarrollo del manual de usuario.

Ciclo de vida de la memoria y defensa del proyecto

- Diseño de la memoria del proyecto.
- Diseño de la defensa del proyecto.
- Desarrollo de la memoria, elaboración de la misma.
- Elaboración de la presentación del proyecto ante el tribunal.
- Entrega de la memoria del proyecto.
- Defensa del proyecto.

2.5.4 Estructura de descomposición de trabajo

Tomando las fases indicadas en el anterior apartado, se obtiene la estructura de la figura 1, donde se observan las distintas tareas que se deben desarrollar para lograr el resultado esperado.

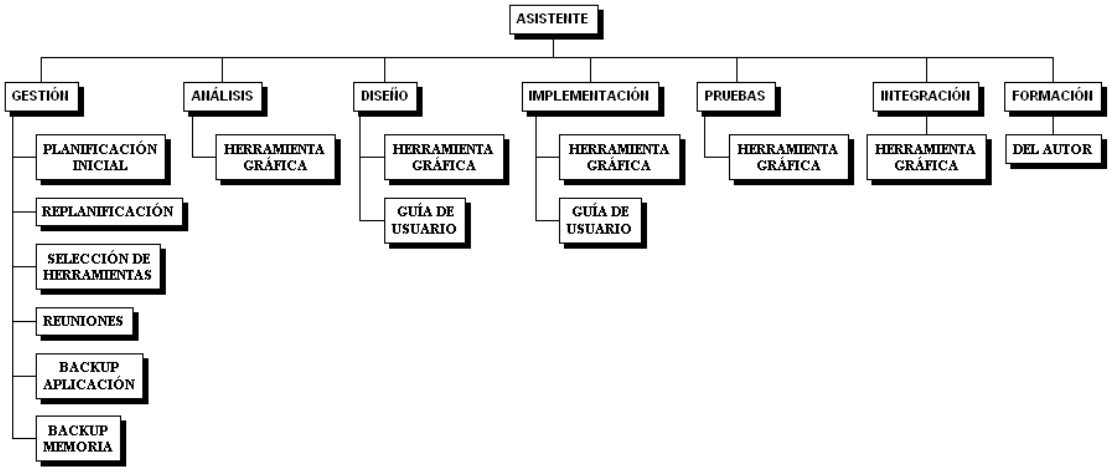

Figura 1: Diagrama EDT

2.5.5 Estimación de horas por tarea

Este apartado muestra las horas estimadas que se asignan a cada tarea. Dado que el proyecto es un proyecto de la carrera de Ingeniería Técnica equiparada a 6 créditos, resultan un total de 180 horas de trabajo. Sin embargo, dada la complejidad del proyecto se llegó a un acuerdo entre el proyectando y el tutor para que se pudiesen incrementar el número de horas. Esto explica el porque el número de horas inicialmente planificadas eran superiores a 180 horas.

Las tablas que se presentan a continuación muestran el reparto de horas que se estiman para el desarrollo del proyecto: la tabla 1 expone el desglose de horas estimadas para los aspectos relativos a la gestión del proyecto; en la tabla 2 se muestran las horas estimadas para el análisis; la tabla 3 muestra las horas previstas para el diseño de la herramienta gráfica y la guía de usuario; la tabla 4 enseña las horas estimadas para la implementación de la herramienta gráfica y guía de usuario; la tabla 5 muestra las previstas para realizar las pruebas a la aplicación; en la tabla 6 las horas de integración en Egoki; la formación del proyectando se presenta en la tabla 7; y finalmente en la tabla 8 podemos observar el cómputo total de horas estimadas para el desarrollo del proyecto.

Proyecto Asistente

1. Gestión

Tarea	Horas
Planificación inicial	20
Reuniones	12
Backup aplicación	1
Memoria proyecto fin de carrera	50
Preparar presentación proyecto	8
	91

Tabla 1: Horas estimadas para la Gestión del proyecto

2. Análisis

Tarea	Horas
Herramienta gráfica	20
	20

Tabla 2: Horas estimadas para el Análisis

3. Diseño

Tarea	Horas
Herramienta gráfica	40
Guía de usuario	1
	41

Tabla 3: Horas estimadas para el Diseño

4. Implementación

Tarea	Horas
Herramienta gráfica	120
Guía de usuario	3
	123

Tabla 4: Horas estimadas para las Pruebas

5. Pruebas

Tarea	Horas
Herramienta gráfica	3
	3

Tabla 5: Horas estimadas para las Pruebas

6. Integración

Tarea	Horas
Herramienta gráfica	2
	2

Tabla 6: Horas estimadas para la integración en Egoki

7. Formación

Tarea	Horas
Del autor	50

Tabla 7: Horas estimadas para la formación del proyectando

Total horas

330

Tabla 8: Horas totales estimadas

2.5.6 Calendario

El calendario laboral inicial que se ha acordado contempla trabajar cinco horas diarias los días laborables, teniendo en cuenta excepciones para los días festivos y motivos personales.

Previamente a la fecha de comienzo, el desarrollador del proyecto ha acordado con su tutor la realización del mismo y ha recibido material de estudio para comenzar a asimilar conceptos y conocimientos necesarios para el desarrollo del mismo.

Fecha de inicio: 5 de Septiembre de 2011

Fecha de fin: 31 de enero de 2012

Días no laborables:

- Octubre: 12
- Noviembre: 3
- Diciembre: 6, 9
- Enero: 6

El acuerdo para realizar el proyecto se ha realizado en el mes de Marzo de 2011, pero se ha pospuesto su comienzo hasta el final de curso en Junio. Durante el verano el desarrollador ha comenzado a estudiar el material didáctico recibido, en este tiempo se han alternado episodios de estudio con episodios de ocio por lo que este período no se ha contabilizado.

La figura 2 muestra el diagrama de Gantt que refleja el tiempo de dedicación previsto para las diferentes tareas planeadas.

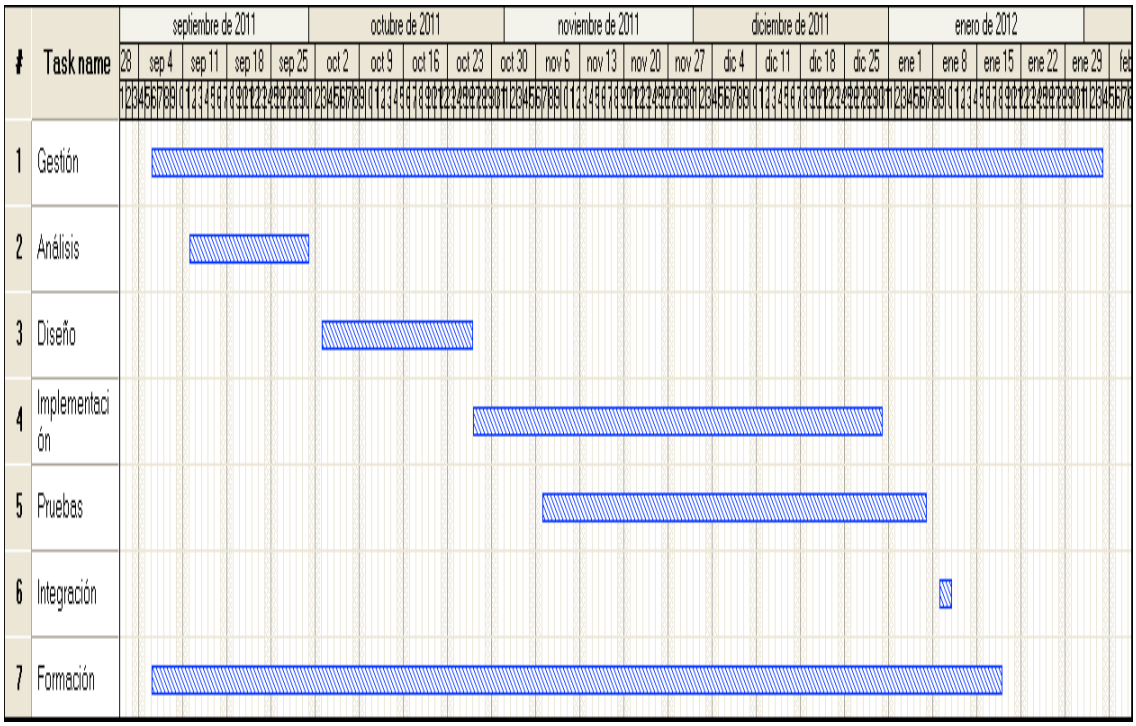

Figura 2: Gantt inicial

2.5.7 Resultado obtenido

En este apartado se compara el tiempo planificado que se puede observar en las tablas del apartado 2.5.5 con el tiempo consumido realmente. En las siguientes tablas se muestra el tiempo real consumido, así la tabla 9 presenta las horas utilizadas en los aspectos referidos a la gestión del proyecto; las tablas 10 y 11 se refieren al análisis y diseño de la herramienta gráfica; y la tabla 12 a la implementación de la misma; las horas consumidas en las pruebas realizadas al asistente y su integración en Egoki se contemplan en las tablas 13 y 14 respectivamente; la formación que se ha necesitado se presenta en la tabla 15; y la tabla 16 expone la suma de horas consumidas.

1. Gestión

Tarea	Horas
Planificación inicial	20
Reuniones	15
Backup aplicación	1
Memoria proyecto fin de carrera	50
Preparar presentación proyecto	8
	94

Tabla 9: Horas consumidas en la Gestión

2. Análisis

Tarea	Horas
Herramienta gráfica	30
	30

Tabla 10: Horas consumidas en el Análisis

3. Diseño

Tarea	Horas
Herramienta gráfica	45
Guía de usuario	1
	46

Tabla 11: Horas consumidas en el Diseño

4. Implementación

Tarea	Horas
Herramienta gráfica	130
Guía de usuario	3
Total horas implementación	133

Tabla 12: Horas consumidas en la Implementación

5. Pruebas

Tarea	Horas
Herramienta gráfica	3
	3

Tabla 13: Horas consumidas en las Pruebas

6. Integración

Tarea	Horas
Herramienta gráfica	2
	2

Tabla 14: Horas consumidas en la Integración en Egoki

7. Formación

Tarea	Horas
Del autor	77
	77

Tabla 15: Horas consumidas en la Formación del proyectando

Total horas
385

Tabla 16: Horas totales consumidas

Comenzado el desarrollo del proyecto con el arranque del mes de Septiembre, durante el mes de Noviembre se realiza una replanificación del proyecto tras comprobar que el tiempo necesario para la formación del proyectando en las técnicas necesarias para construir la herramienta gráfica, era bastante superior al inicialmente estimado.

La situación en el momento de la replanificación indicaba un desfase con respecto a lo planeado, debido al retraso en el análisis y el diseño que debían estar realizados. Este retraso era debido fundamentalmente a lo relatado en el anterior párrafo, también el panorama próximo de fechas festivas por la navidad condicionó la decisión de replanificar el tiempo necesario para la correcta finalización del proyecto.

Se acuerda prolongarlo hasta semana santa lo que aumenta la estimación inicial hasta mediados de Abril, es decir dos meses más.

Esta replanificación y la posterior distribución de las horas consumidas se pueden ver en la figura 3.

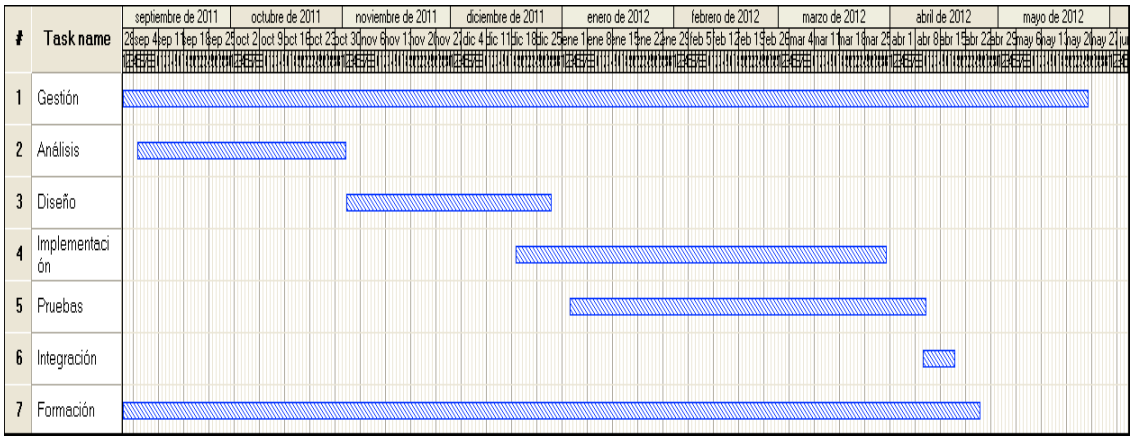

Figura 3: Gantt final

Considerando los datos recogidos durante el seguimiento del proyecto y los cambios que ha supuesto la replanificación del mismo, se puede deducir el porcentaje de tiempo dedicado a cada uno de los aspectos más relevantes, de la creación de la herramienta

gráfica. En la figura 4 se destacan los porcentajes de tiempo dedicado al estudio de EMF y GMF, el desarrollo de la herramienta gráfica, el tiempo necesario para documentación y la creación de la memoria del proyecto.

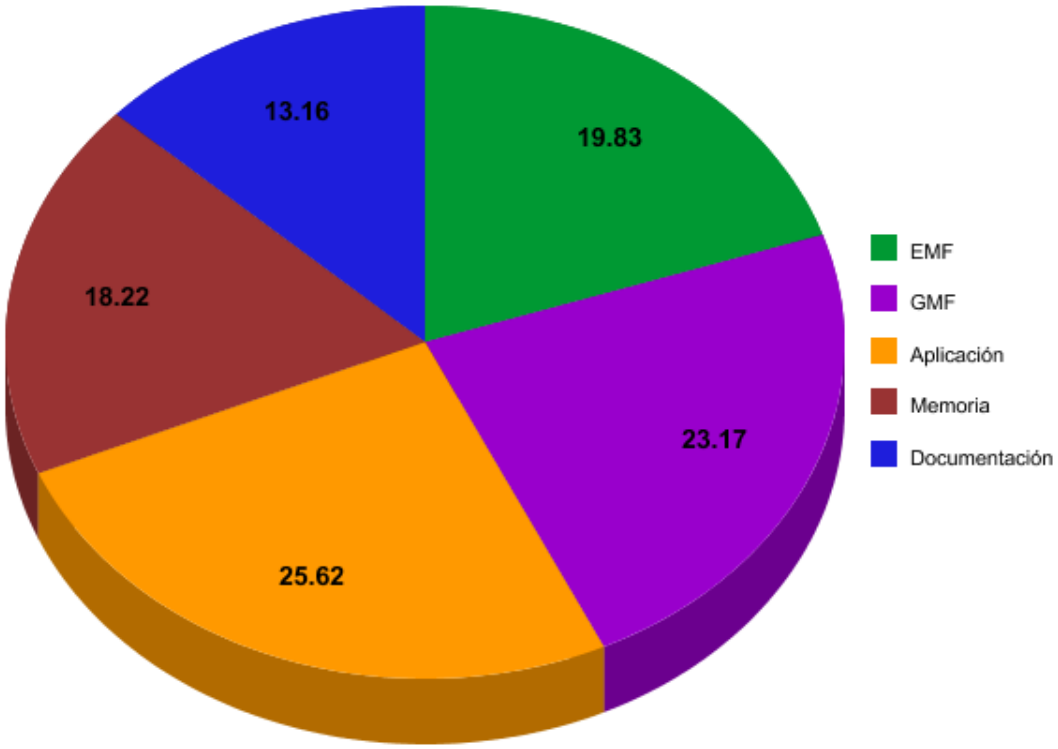

Figura 4: Porcentaje de tiempo

● EMF (19.83%): Aprendizaje de los conceptos del desarrollo de software dirigido por modelos, estudio de la herramienta Eclipse Modeling Framework y creación del metamodelo.

● GMF (23.17%): Estudio de la herramienta Graphical Modelling Framework de Eclipse y creación del editor para la composición de la interfaz abstracta de usuario.

● Aplicación (25.62%): Creación y pruebas de funcionamiento e integración de la herramienta gráfica, usando los proyectos EMF, GMF y RCP de Eclipse.

● Memoria (18.22%): Recopilación del material necesario para crear la memoria del proyecto. Elaboración de la memoria.

● Documentación (13.16%): Tiempo invertido en buscar la información necesaria para el análisis y diseño de la aplicación. Además del tiempo invertido en la búsqueda del material, que se ha necesitado para el estudio de los distintos aspectos del proyecto.

2.5.8 Concordancia entre planificación y resultados

Analizando estos datos, se puede observar que la mayor cantidad de tiempo se ha invertido en aspectos relativos al estudio de las tecnologías EMF y GMF. Para la recopilación de datos, estudio y aprendizaje de estos dos elementos se ha encontrado el inconveniente de la falta de información sobre los mismos y eso ha hecho que se haya necesitado más tiempo del previsto para aprenderlos y utilizarlos en el desarrollo de la herramienta gráfica.

Otro aspecto que se quiere resaltar, es que el uso del IDE Eclipse para el desarrollo de software dirigido por modelos es muy recomendable, pero su arquitectura requiere una curva de aprendizaje elevada para conocerla e integrar los módulos necesarios que nos permiten desarrollar la herramienta gráfica que se ha creado.

Si analizamos el trabajo efectuado mes a mes, podemos observar que tras la replanificación efectuada a finales de Noviembre, aumenta el volumen de horas invertidas en el proyecto debido a la necesidad de conocer mejor la tecnología necesaria para el desarrollo del producto.

A principios de Diciembre, comienza a consumirse una mayor cantidad de horas que se corresponden con los meses donde se incrementa el esfuerzo en el aprendizaje de EMF y GMF. Además habituarse a la estructura de RCP también influyó con este incremento. Este período de adaptación a las herramientas y de asentamiento de la implementación concluye en el mes de Febrero.

A partir de Marzo baja el flujo de trabajo con respecto al período anterior, pero se mantiene un alto nivel de trabajo coincidiendo con los aspectos finales de implementación, pruebas del sistema e integración con Egoki. Tras semana santa baja notablemente el número de horas debido a las dos semanas que suponen las vacaciones, el flujo de trabajo se traslada hacia la recopilación de información para la memoria, creación de la misma y otros ajustes finales de la herramienta gráfica. En la figura 5 se muestra la distribución de las horas empleadas en el desarrollo del PFC.

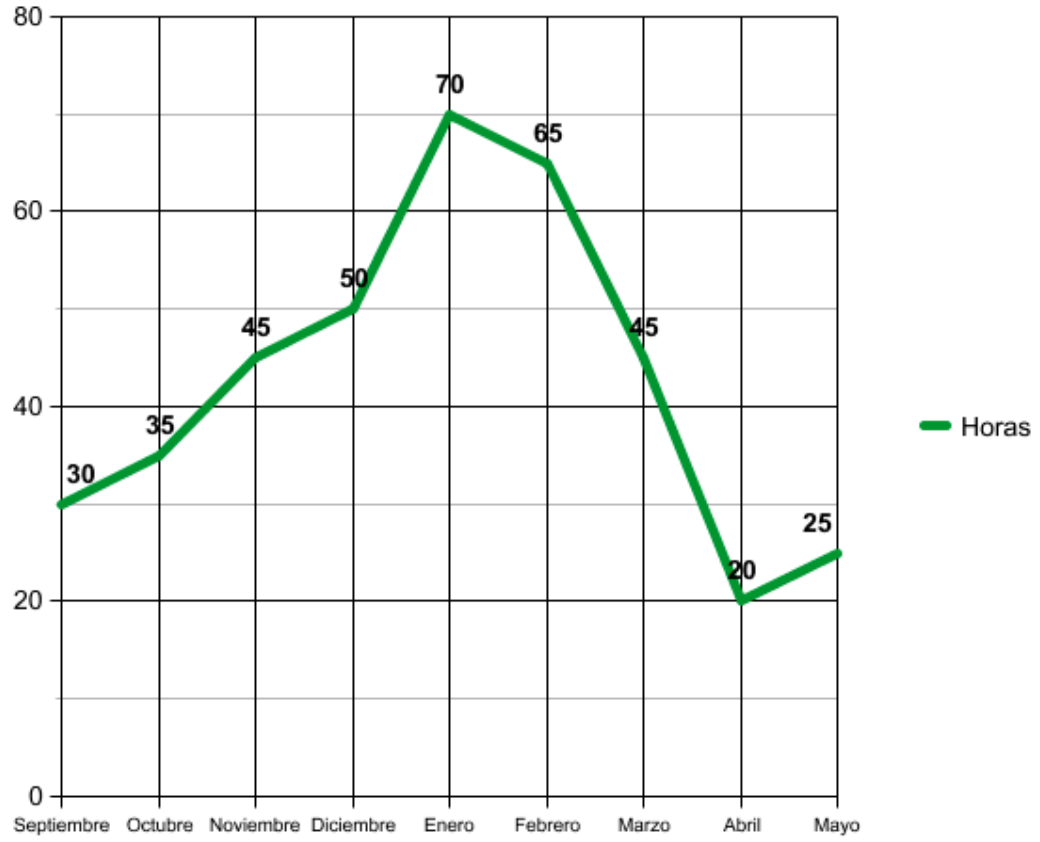

Figura 5: Distribución de horas

Finalmente, el proyecto ha sufrido un retraso con respecto a la planificación inicial que se estimaba para finales de Enero, habiéndose alargado más tiempo de lo inicialmente previsto.

2.6 Riesgos

● Hacer una estimación errónea de la planificación temporal.

● Pérdida de información. Mala gestión de las copias de seguridad y del archivo de información.

● No finalizar la aplicación en el plazo acordado con la consiguiente desmotivación del autor y del equipo que le apoya.

● Falta de comunicación y descoordinación entre el autor del proyecto y el equipo que dirige el proyecto.

2.7 Oportunidades

● Integración en Egoki.

● Aprendizaje de la tecnología EMF.

● Aprendizaje de la tecnología GMF.

● Aprendizaje de la tecnología RCP.

● Elaboración de un proyecto de ingeniería informática.

2.8 Partidos interesados en el proyecto

En este apartado se identifican las personas o colectivos a las que puede afectar de algún modo la elaboración de este proyecto, son los partidos interesados en el mismo:

● Desarrollador del proyecto.

● Tutor del proyecto.

● Diseñador de servicios ubicuos.

2.9 Expectativas de los partidos interesados

Desarrollador

● Finalizar el proyecto en el plazo establecido para poder defenderlo y terminar su trayectoria universitaria.

● Realizar un producto consistente, robusto y de calidad.

● Integrar el proyecto en Egoki.

● Ofrecer un producto que sea del agrado del resto de implicados.

● Ofrecer una imagen de implicación y profesionalidad.

Tutor

● Guiar al alumno en el desarrollo de un proyecto de software, estimulándolo para organizarse y superar por si mismo los obstáculos que se pueda encontrar durante el progreso del proyecto.

● Orientar al alumno para el buen fin de su trabajo.

● Conseguir que el trabajo del alumno tenga el nivel de calidad exigido.

● Aumentar el nivel de conocimiento del alumno sobre el desarrollo de un producto software.

● Conseguir que el trabajo de tutorización no suponga una carga extra de trabajo que dificulte el normal desarrollo de sus tareas habituales.

Diseñador

● Obtener una herramienta gráfica que le permita construir interfaces abstractas de usuario de una forma sencilla.

● Que la herramienta sea sencilla de manejar.

● Evitar la necesidad de tener conocimientos sobre XML.

● Disponer de una descripción de la interfaz abstracta sin necesidad de escribir un archivo UIML.

● Ahorrar tiempo en la descripción de interfaces abstractas de usuario para servicios ubicuos.

● Disponer de las herramientas necesarias para desarrollar el servicio ubicuo diseñado.

2.10 Costes

En este apartado se desarrolla el plan de costes del proyecto. Se ha realizado la estimación de costes según el juicio experto del autor, valorando a este como programador junior, cobrando 15.000 €/año brutos, lo que supone 21 €/hora. Al tutor de proyecto se le ha estimado una tarifa de 50 €/hora.

El software proveniente de la empresa Microsoft se ha adquirido de forma gratuita, ya que para usos derivados de la formación la empresa tiene un convenio firmado con la Facultad.

El IDE Eclipse y los proyectos EMF, GMF y RCP, son herramientas de código abierto y su uso es libre, lo que no supone un aumento del coste del proyecto.

El resto de aplicaciones son versiones libres o de prueba que no implican coste adicional en el proyecto.

En la tabla 17 se expone el desglose de costes del proyecto.

Recurso	Coste/Uso	Unidades	Coste Total
Desarrollador	21 €/hora	519 horas	10899 €
Tutor	50 €/hora	20 horas	1000 €
Ordenador AMD Turion	700 €/unidad	1 unidad	700 €
Línea ADSL 10Mb	36 €/mes	5 meses	180 €
Eclipse 3.6 Helios	0 €/unidad	1 unidad	0 €
EMF	0 €/unidad	1 unidad	0 €
GMF	0 €/unidad	1 unidad	0 €
RCP	0 €/unidad	1 unidad	0 €
DropBox	0 €/unidad	1 unidad	0 €
Dia	0 €/unidad	1 unidad	0 €
Microsoft Office Word 2003	0 €/unidad	1 unidad	0 €
Total			12779 €

Tabla 17: Coste del proyecto

2.11 Análisis de Factibilidad

Tras establecer los objetivos del proyecto, puede realizarse una valoración global sobre las posibilidades de realizar y desarrollar el trabajo encomendado de forma satisfactoria.

Valorando inicialmente los componentes tecnológicos necesarios para la correcta creación de la herramienta se hace un análisis de las tecnologías requeridas que se han presentado en el apartado 2.5.1 y se llega a la conclusión de que estos cubren las necesidades para poder desarrollar una aplicación estable, cubriendo el alcance fijado.

Al estudiar los componentes tecnológicos finalmente seleccionados y otros que posteriormente se han descartado, se ha podido comprobar que existen varios de ellos que permiten desarrollar la aplicación que se pretende conseguir e incluso ofrecen funcionalidades adicionales a las necesarias. Por ello, se concluye que tras conocer el trabajo a desarrollar y habiendo analizado y escogido los componentes tecnológicos necesarios para realizar dicho trabajo el presente proyecto es factible.

CAPÍTULO 3 ESTUDIO DE ANTECEDENTES

Se han desarrollado distintas herramientas gráficas para generar diferentes documentos de Lenguajes Descripción de Interfaces de Usuario (UIDL) [4]. Varias de ellas para el lenguaje UsiXML [8], de las que cabe destacar:

● IDEALXML [11] es una herramienta que proporciona un editor simple utilizado por diseñadores de aplicaciones para compartir conocimiento. Esta herramienta permite editar modelos de tareas (basados en la herramienta CTTE [13]), interfaces abstractas de usuario, modelos de dominio y modelos de mapeo.

● SketchiXML [2] es una aplicación que permite manejar diferentes tipos de fuentes hechas a mano como entrada y proporcionar la correspondiente especificación UsiXML.

● GUILayout++ [12] es una herramienta para diseñar interfaces centradas en el usuario a través de un proceso iterativo basado en el prototipado y la evaluación. Es capaz de generar automáticamente interfaces abstractas de usuario con la sintaxis UsiXML desde el prototipo creado.

Con la sintaxis UIML también se ha desarrollado varias herramientas, como por ejemplo Liquid Apps [7]. Esta herramienta permite editar gráficamente interfaces abstractas de usuario y generar el código UIML correspondiente. Por otra parte, GUMMY [10] es una aplicación para construir interfaces multi-plataforma gráficamente, basándose en UIML.

Todas estas herramientas han sido analizadas y tenidas en cuenta a la hora de realizar el presente proyecto, sin embargo en ninguna se han encontrado implementadas las funcionalidades de generar la interfaz abstracta de usuario a partir de la descripción de un servicio, ni la funcionalidad de asociar recursos multimedia a los elementos abstractos de interacción. Por ello, se ha visto la necesidad de elaborar el presente proyecto.

CAPÍTULO 4 DESARROLLO TÉCNICO

Este capítulo esta dedicado a explicar como se han llevado a cabo las diferentes fases que componen el ciclo de desarrollo del proyecto: captura de requisitos, descripción de las funcionalidades del sistema, implementación de la herramienta. También se expone el estudio tecnológico realizado y se muestran las interfaces creadas.

4.1 Captura de requisitos

● Permitir que el diseñador seleccione los archivos de servicio para proporcionárselos a la herramienta.

● Ofrecer al diseñador una paleta de herramientas que le permitan estructurar una interfaz abstracta de usuario.

● Permitir que el diseñador proporcione los archivos que contienen los recursos del servicio.

● Mostrar los objetos abstractos de interacción que componen la interfaz creada.

● Permitir su selección.

● Mostrar los recursos que pueden ser asociados a los objetos abstractos de interacción.

● Permitir la asociación entre el objeto abstracto de interacción seleccionado y los recursos disponibles.

● El diseñador puede en cualquier momento deshacer la asociación y volver a elegir un objeto abstracto de interacción.

● Generar un documento UIML que muestre la estructura de la interfaz abstracta de usuario y los recursos asociados a sus componentes.

4.2 Estudio Tecnológico

En este capítulo se enumeran las tecnologías principales que se han estudiado para llevar a cabo el presente proyecto y se ofrece una breve descripción de cada una de ellas.

4.2.1 Ingeniería dirigida por modelos

La ingeniería dirigida por modelos surge como la respuesta de la ingeniería de software a la industrialización del desarrollo de software. MDA (Model Driven Architecture) es la propuesta de la OMG (Object Manage Group) [14], que centra sus esfuerzos en reconocer, que la interoperabilidad es fundamental y que el desarrollo de modelos permite la generación de otros modelos que posteriormente al combinarse puedan proveer la solución a todo un sistema, independizando el desarrollo de las tecnologías empleadas.

Un metamodelo define los elementos de un lenguaje de modelado, las relaciones entre ellos y las restricciones. Define la sintaxis abstracta y la semántica estática pero no la sintaxis concreta.

4.2.1.1 Eclipse Modelling Framework: EMF

El proyecto Eclipse Modelling Framework (EMF) [15] proporciona un marco de modelado que facilita la generación de código para construir herramientas y otras aplicaciones basadas en un modelo de datos estructurado. Desde una especificación del modelo descrita en el lenguaje XML Metada Interchange (XMI), EMF proporciona las herramientas necesarias y el soporte de ejecución para producir un conjunto de clases Java a partir del modelo.

Debido a que las descripciones UIML son documentos estructurados se ha utilizado EMF para conceptualizar esta sintaxis, ya que nos permite crear un metamodelo de la interfaz abstracta de usuario que posteriormente es usado como origen para desarrollar el editor.

4.2.1.2 Graphical Modelling Framework: GMF

El proyecto Eclipse denominado Graphical Modelling Framework (GMF) [3] proporciona un conjunto de componentes generadores e infraestructuras en tiempo de ejecución para el desarrollo de editores gráficos basados en EMF y Graphical Editing Framework (GEF) [15]. Por ello, se ha escogido esta tecnología para generar el editor gráfico de documentos UIML, a partir del meta-modelo construido con EMF.

GMF ofrece un enfoque basado en modelos para la generación de editores gráficos en Eclipse. Para ello, se debe definir cuales serán los elementos y las características que se integrarán en el editor de las descritas en el meta-modelo creado a través de EMF; la paleta de herramientas a utilizar y el mapeo entre los anteriores elementos definidos. En la figura 6 se muestran el flujo de pasos que se debe seguir para construir un editor gráfico.

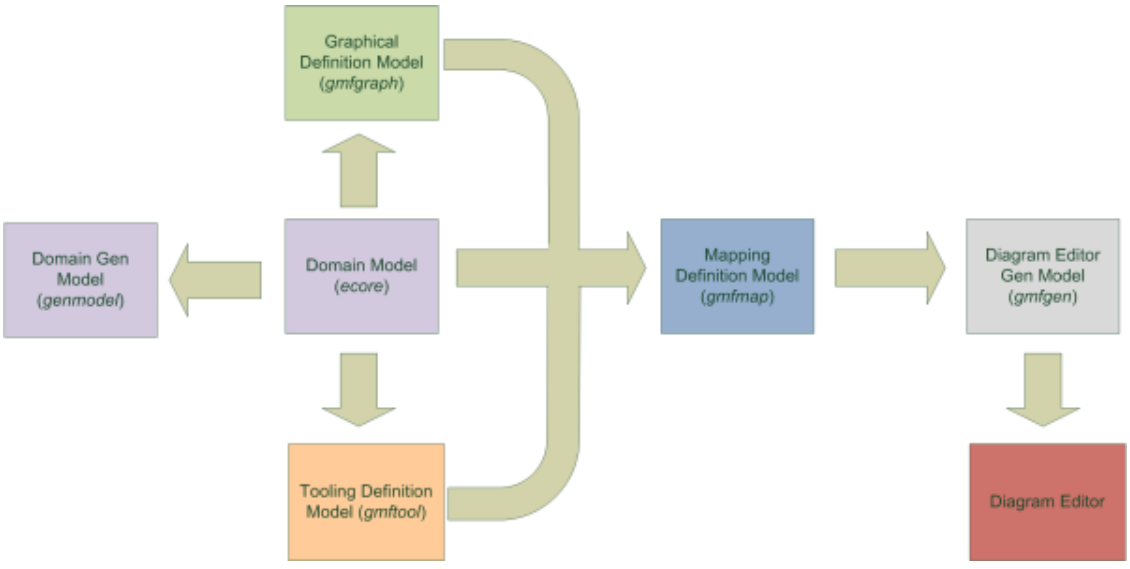

Figura 6: Flujo para crear una aplicación con GMF

Por otra parte, en la figura 7 se ilustra el asistente que proporciona el entorno GMF para guiar al desarrollador en el flujo de GMF.

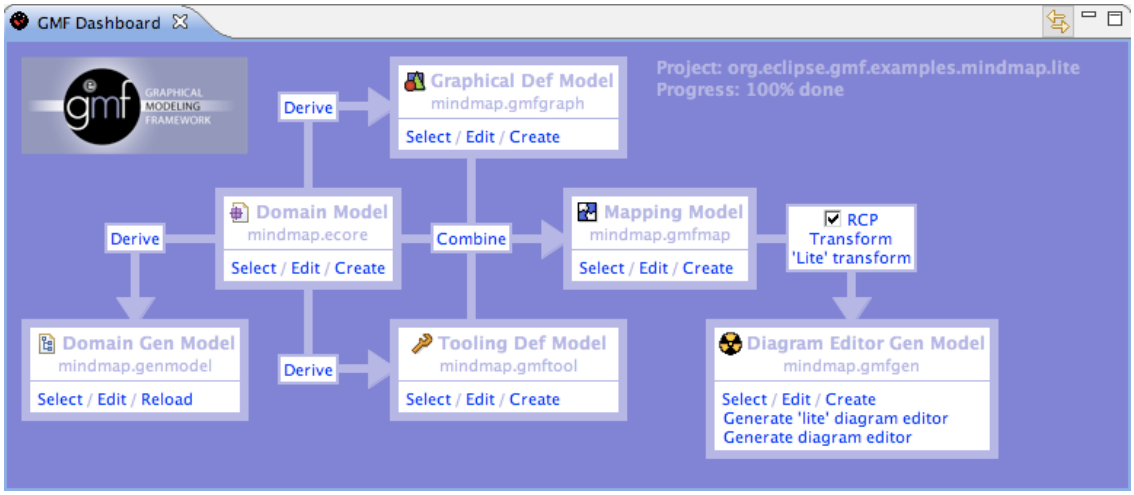

Figura 7: Asistente de Eclipse GMF para crear un editor

Como se ha comentado anteriormente, el primer paso es generar el modelo del dominio a través de EMF (.ecore). En este modelo de dominio se indicará que elementos serán representados de manera visual, sus relaciones y propiedades. Posteriormente se deben generar las clases Java relativas al modelo (.genmodel).

Después se debe definir el modelo gráfico que tendrá el editor (.gmfgraph). En este modelo se definen los elementos gráficos para representar los elementos del dominio en el editor gráfico. Se definen los elementos visuales que serán desplegados en el editor, como son los nodos, enlaces, figuras, etiquetas, etc. Se puede ver un ejemplo de este modelo en la figura 8.

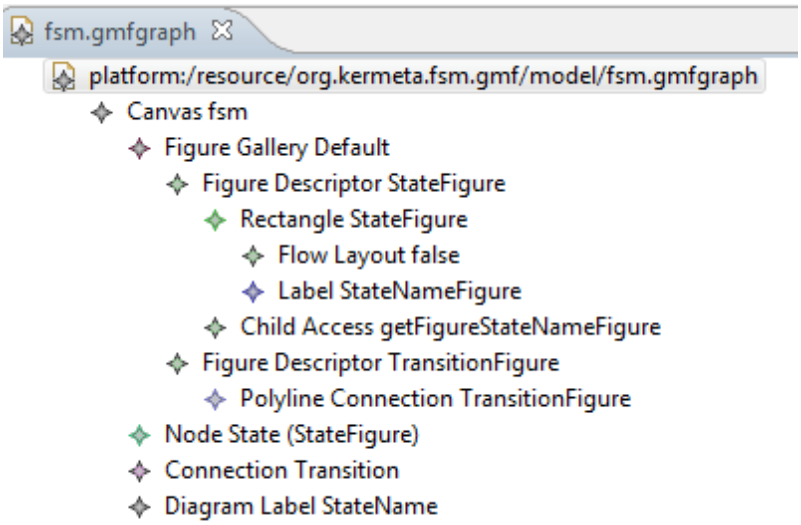

Figura 8: Ejemplo de fichero .gmfgraph

En el fichero .gmftool se define el modelo de herramientas que estarán presentes en el editor. Se refiere a los elementos que estarán disponibles en la paleta de herramientas que tendrá el editor. Representa los elementos visuales de dibujo que tendrá el editor tales como menús, botones, etc. La figura 9 ilustra un ejemplo de este modelo.

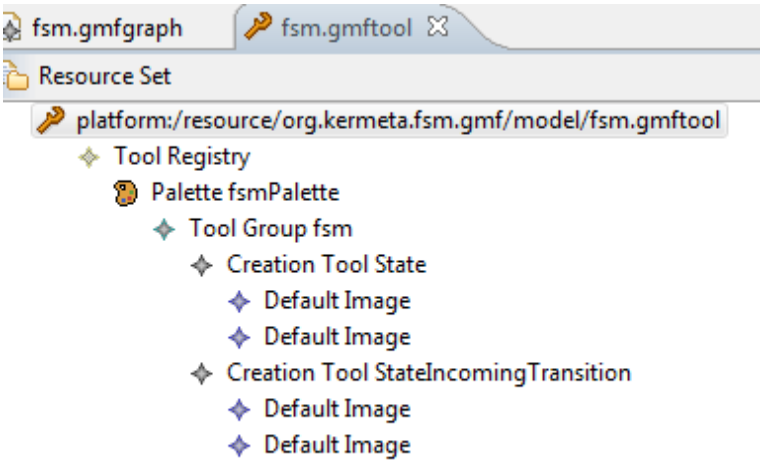

Figura 9: Ejemplo de fichero .gmftool

En este punto del proceso se tienen tres modelos: del dominio, de definición gráfica y de definición de la paleta de herramientas. Para unir todos los modelos y definir qué elementos corresponden a qué elementos entre los distintos modelos es necesario crear un modelo que una todos los conceptos representados en los tres anteriores. Este es el modelo de mapeo (.gmfmap), que utiliza los elementos del modelo de dominio, asignándoles una figura del modelo gráfico y el elemento del modelo de la definición

gráfica de la paleta con la que se dibujarán. Se puede ver una muestra de este modelo en la figura 10.

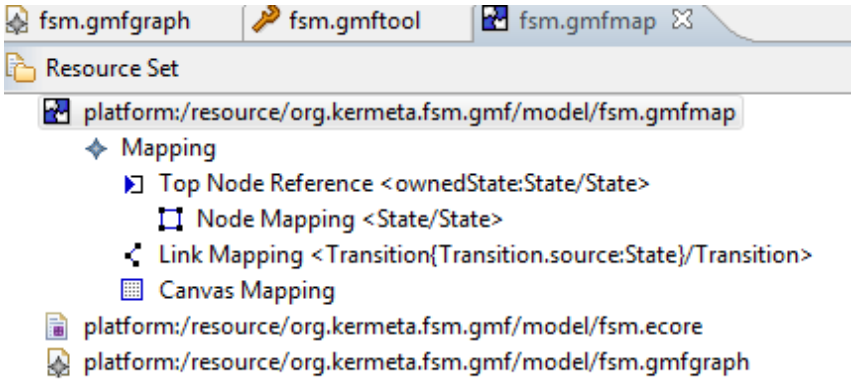

Figura 10: Ejemplo de fichero.gmfmap

Una vez que se han definido los cuatro modelos básicos, mediante GMF se genera el llamado modelo generador (Figura 11).

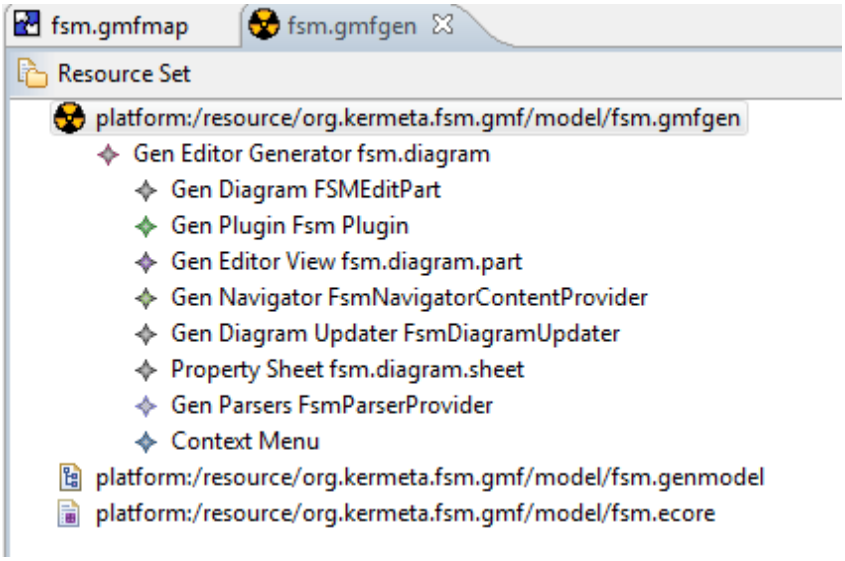

Figura 11: Ejemplo de fichero .gmfgen

Este modelo es generado por GMF a partir del modelo de mapeo y contiene toda la información de los cuatro modelos anteriores, además de toda la información necesaria para crear el código del editor. Los modelos que serán creados con el editor gráfico serán guardados con formato XML. En este modelo generador se puede definir la extensión que tendrán estos archivos de manera que el editor generará dos archivos por modelo, uno con extensión .diagram que almacena los elementos visuales del modelo y

otro con extensión .model en formato XMI que contiene los elementos del dominio. Cuando se genera este código se crea una aplicación de tipo Rich Client Platform (RCP) para ejecutar el código.

4.2.2 Eclipse RCP: Rich Client Platform

El editor gráfico que se genera a través de GMF es del tipo Rich Client Platform (RCP) [9] [16]. RCP es un subconjunto de plug-ins de Eclipse que nos permite desarrollar cualquier tipo de aplicación, aprovechando las características más potentes de Eclipse.

Eclipse RCP es un Framework a partir del cual podemos construir aplicaciones sobre la base de Eclipse. RCP proporciona un Eclipse "vacío" que podemos modificar para crear diferentes tipos de aplicaciones explotando los mecanismos de perspectivas, vistas o asistentes.

4.2.3 User Interface Modelling Language: UIML

El metamodelo que se ha desarrollado con EMF representa la estructura de un documento UIML y en nuestro caso se ha decidido crearlo en su totalidad aunque para el editor en concreto no era necesario. Se ha decidido así porque de esa manera teniendo el metamodelo en su totalidad nos sirve para otros módulos necesarios en la creación de la herramienta gráfica.

En la figura 12 se puede ver el metamodelo simplificado de un documento UIML, no es realmente el metamodelo usado en la construcción del editor puesto que debido a su gran extensión no es posible integrarlo en este documento con un grado aceptable de visión, pero refleja perfectamente la estructura básica de un documento UIML.

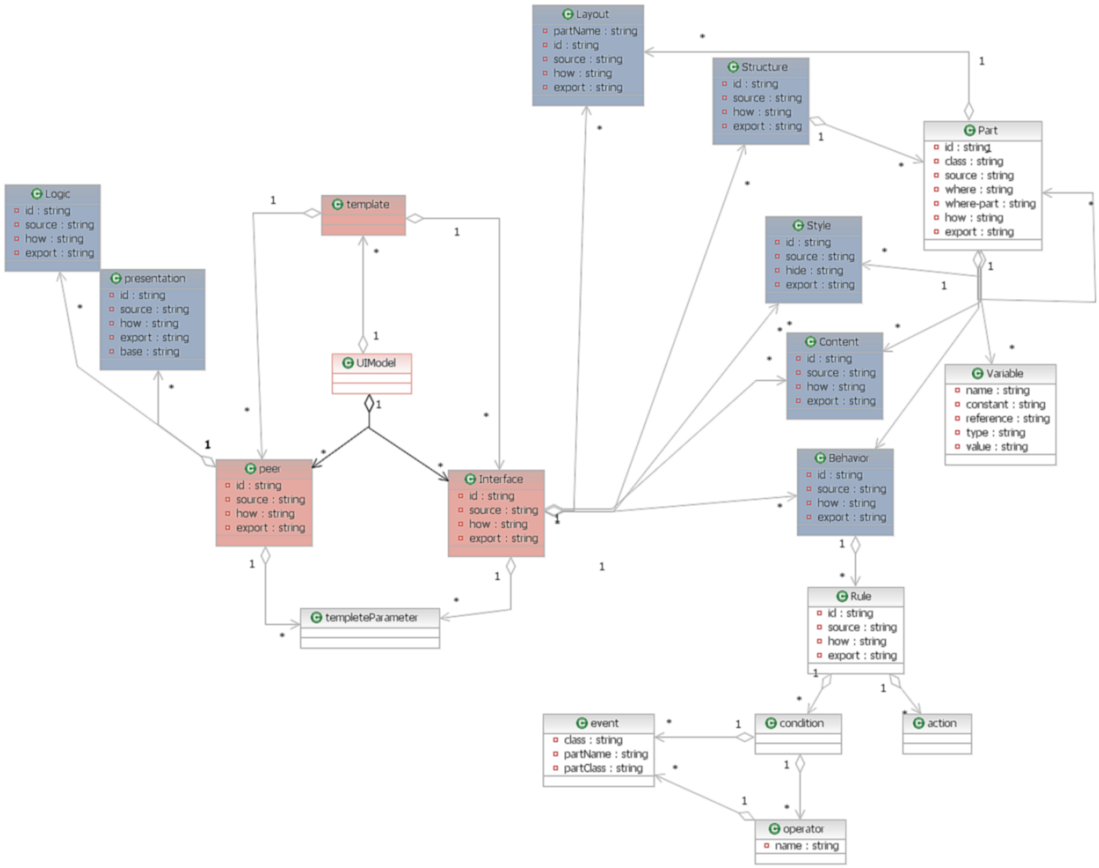

Figura 12: Metamodelo simplificado de UIML

Un documento UIML consta en un primer nivel de varios elementos que especifican distintas partes del servicio, en nuestro caso el elemento que interesa es el elemento "Interface" que es el que representa una posible manera de renderizar la interfaz de usuario. De los distintos elementos que forman el siguiente nivel el elemento "Structure", que indica la estructura de la interfaz de forma abstracta, es el que contiene a los elementos concretos que forman la interfaz abstracta, los elementos "Part". El elemento "Part" identifica los elementos de interacción abstractos y describe una parte específica de la interfaz. Estos elementos son los que realmente modelamos para el editor gráfico y son los que forman la paleta de opciones del editor creado. Se han identificado las distintas representaciones posibles para un elemento "Part" y en nuestra paleta de herramientas se representaran los siguientes:

1. Contenedor: Elemento contenedor de otros elementos "Part".
2. Desplegable: Elemento de interacción que contiene elementos de tipo "Item".
3. Item: Elemento del desplegable.
4. Botón: Elemento disparador de acciones.
5. Input: Elemento que proporciona información de entrada.
6. Boolean: Elemento de entrada que ofrece sólo dos opciones.
7. Salida: Elemento que muestra información y no se puede interactuar con él.

Otros elementos del documento UIML del nivel siguiente al del elemento “Interface” que son necesarios conocer son “Style”, elemento donde se relacionan propiedades con los elementos “Part”, el elemento “Content” que establece el contenido de los elementos “Part” y el elemento “Behavior” que consta de una serie de reglas condición-acción que definen lo que sucede cuando un usuario interactúa con un elemento de la interfaz de usuario. Otro elemento interesante es el elemento “Peers”, donde se mapean las clases, propiedades y eventos con entidades externas al documento.

Los elementos “Style” y “Content” son necesarios en esta herramienta gráfica para identificar el contenido y los recursos asociados de los elementos “Part”. Estos elementos se utilizan en todo lo relacionado con la selección de archivos y la asociación de recursos.

4.2.4 XML

La tecnología eXtensive Markup Langauge (XML) [20] esta diseñada para transportar y almacenar datos. XML es un lenguaje textual estructurado sobre el que, entre otras funciones, se permite describir una sintaxis. En nuestro contexto se ha decidido estudiar esta tecnología debido a que los documentos UIML y los documentos que describen los servicios ubicuos están basados en XML.

Para parsear, modificarlos y crear, este tipo de documentos se ha utilizado la librería Java Apache Xerces [18].

4.2.5 XSL

XSL (eXtensible Stylesheet Language Family) [19] es un conjunto de recomendaciones para definir transformaciones y presentaciones sobre documentos XML. Dentro de esta familia se define como hacer las transformaciones con la sintaxis XSLT (XSL Trasnformations). Se ha utilizado la librería Java Apache Xalan [17] para transformar las descripciones de los servicios ubicuos a la sintaxis UIML.

4.3 Funcionalidades del sistema

En este apartado se van a explicar las diferentes funcionalidades que puede realizar el diseñador del servicio a través de la herramienta. Este apartado va a ser organizado por los diferentes casos de uso que componen el asistente y además se va a mostrar la representación de sus diagramas de secuencia correspondientes para ilustrar al lector el como se ha hecho.

Inicialmente, en la figura 13, se muestra el caso de uso general donde aparecen representadas todas las funcionalidades, a continuación en los siguientes sub-apartados se entrará a explicar en detalle cada funcionalidad.

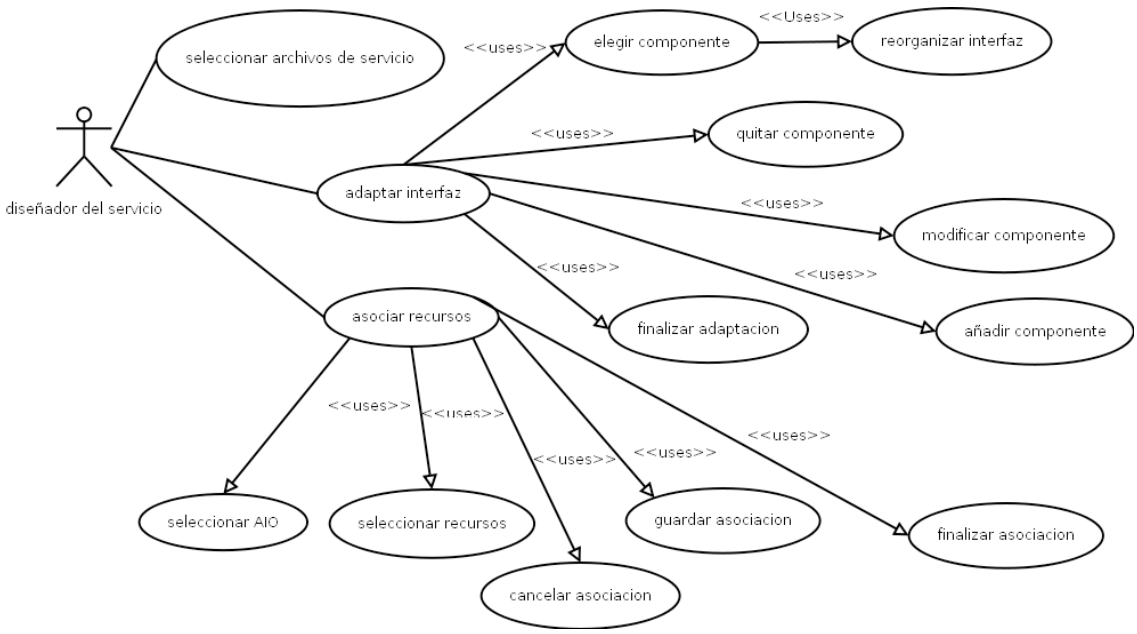

Figura 13: Diagrama de casos de uso

4.3.1 Seleccionar archivos de servicio

Este caso de uso permite al diseñador del servicio seleccionar un archivo en el que se describen las funcionalidades del servicio ubicuo a utilizar. Una vez seleccionado se extrae la información necesaria para generar un documento UIML básico compatible con el editor. Posteriormente se carga en el editor, permitiendo al usuario ejecutar el caso de uso de Adaptar Interfaz.

4.3.2 Adaptar Interfaz

El caso de uso "Adaptar Interfaz" y los casos de uso incluidos en él se refieren al editor gráfico. El editor gráfico se ha creado siguiendo las pautas de la ingeniería dirigida por modelos. Partiendo de un metamodelo donde se definen los conceptos a modelar y sus relaciones se permite la creación, en nuestro caso, de un editor gráfico a partir de dicho metamodelo.

Las funcionalidades del editor gráfico, deben permitir al diseñador *adaptar* a su estilo la interfaz presentada por el sistema. Debe poder *reorganizarla* si lo estima oportuno. Los cambios deben reflejarse en un documento UIML, que el sistema genere para guardar las características de esa interfaz abstracta de usuario para el servicio añadido.

Para permitir la funcionalidad anterior, el diseñador debe poder *seleccionar* los componentes, *quitarlos*, *añadirlos, modificarlos, reorganizarlos* y *finalizar* la adaptación de la interfaz abstracta cuando lo crea conveniente.

El metamodelo que se ha desarrollado con EMF, representa la estructura de un documento UIML y, en nuestro caso, se ha decidido crearlo en su totalidad aunque para el editor en concreto no era necesario. Se ha decidido así porque de esa manera teniendo

el metamodelo en su totalidad nos sirve para otros módulos necesarios en la creación de la herramienta gráfica y para dejar abierta la posibilidad de crear en un futuro nuevos editores más extensos sobre este metamodelo.

El editor gráfico para ofrecer las funcionalidades requeridas, muestra una paleta de herramientas que posibilita al diseñador realizar las acciones que le interesen. Recibe un fichero de entrada como se describe en el apartado 4.3.1. Estos ficheros son ficheros de tecnología XML y sintaxis UIML, por lo tanto estructurados. Eso nos permite utilizar la tecnología EMF de Eclipse, para modelar el fichero de entrada donde se destaquen los elementos "Part" que componen la estructura de la interfaz y que son los que ofrece el editor en su paleta de herramientas.

● Metamodelo: En la figura 12 del apartado 4.2.3 se puede ver el metamodelo simplificado de UIML. Fijándonos en el elemento "Part" de esa figura, que identifica a los elementos de interacción abstractos, se ha desarrollado una estructura de herencia donde la raíz es el elemento "Part" y se han creado tantos elementos hijo, para presentar en la paleta del editor gráfico, como tipos de AIO que queremos tratar del fichero UIML.

● Herencia del elemento Part: a continuación se enumeran los elementos creados en el metamodelo para representar los distintos componentes de la paleta de herramientas del editor gráfico. Se dividen en elementos contenedores y elementos de interacción, estos a su vez se dividen en elementos de entrada, elementos de salida y accionadores.

Elementos Contenedores
Contenedor: Elemento contenedor de otros elementos Part.

Elementos de Interacción
1. Elementos Entrada
   1.1 Desplegable: Elemento de interacción que contiene elementos de tipo Item.
   1.2 Item: Elemento del desplegable.
   1.3 Entrada: Elemento que proporciona información de entrada.
   1.4 Booleano: Elemento de entrada que ofrece sólo dos opciones.
2. Elementos Salida
   2.1 Salida: Elemento que muestra información y no se puede interactuar con él.
3. Elementos Accionadores
   3.1 Botón: elemento que representa un botón abstracto.

La figura 14 ilustra el metamodelo creado para el editor gráfico.

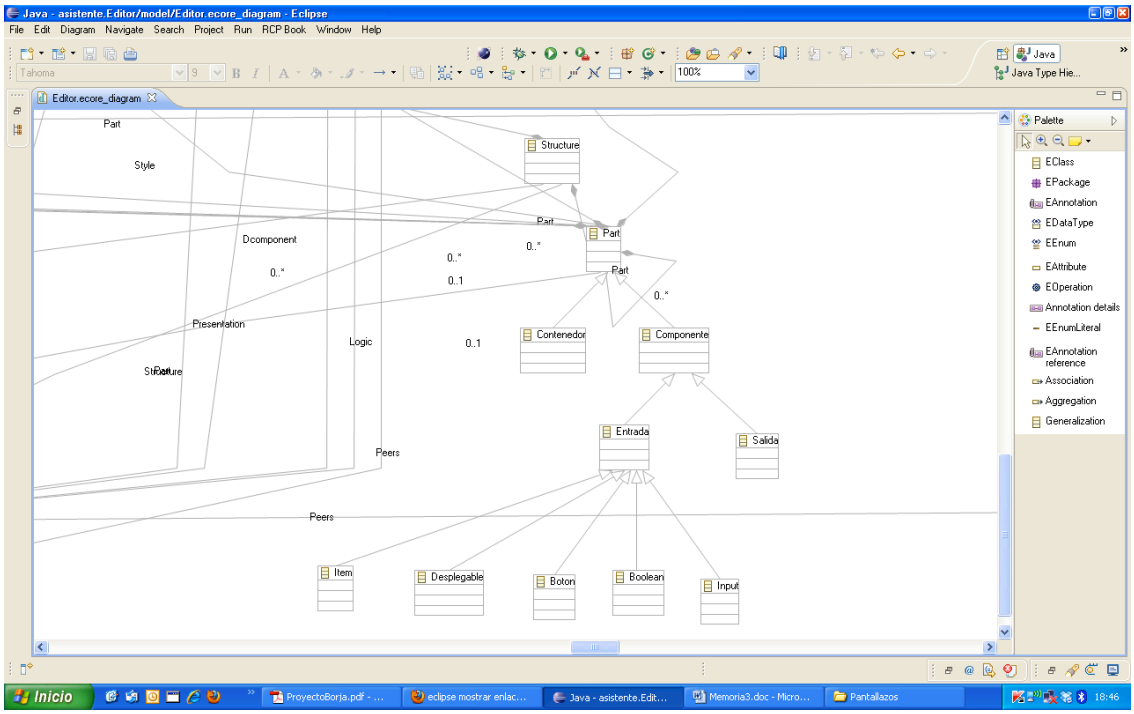

Figura 14: Imagen del metamodelo creado

4.3.3 Asociar Recursos

El diseñador asocia los componentes de la interfaz gráfica que ha creado, en el editor gráfico, con los recursos disponibles. Para permitir la asociación, se ha pensado en una vista que ofrece automáticamente una lista de los AIO incorporados a la interfaz abstracta de usuario, esa vista se denomina VistAIO. Al seleccionar uno de los elementos, se lanza otra vista denominada VistaRecursos, que muestra el nombre del AIO seleccionado y los recursos disponibles que pueden asociarse a ese AIO. Este caso de uso incluye a otros casos de uso y su resultado depende de los resultados de los casos de uso incluidos. En los siguientes subapartados, se exponen los casos de uso incluidos en este.

4.3.3.1 Seleccionar Abstract Interaction Object (AIO)

El diseñador del servicio selecciona un componente de la interfaz gráfica de la tabla que le ofrece el sistema. Al seleccionar un elemento de la tabla el sistema ofrece una lista en forma de árbol con los recursos que se pueden asociar al AIO seleccionado. El AIO seleccionado se guarda en la estructura de salida de datos. Este escenario se contempla en la figura 15.

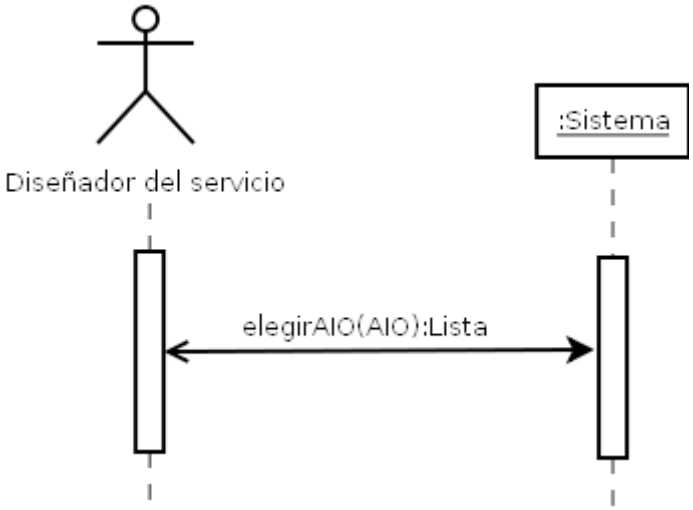

Figura 15: Diagrama de secuencia del sistema. Caso de uso "Seleccionar AIO"

A continuación se realiza una breve descripción de los métodos planificados para este caso de uso y se muestra el diagrama de secuencia del mismo en la figura 16.

El método "guardar" de la clase VistaRecursos recoge el AIO seleccionado y lo guarda para asociarlo con sus recursos. El método "getElements" de la clase Proveedor agrupa y envía los recursos disponibles. El método "crearArbol" de la clase VistaRecursos crea un árbol donde se muestran los elementos de la lista de recursos que le ha enviado el proveedor.

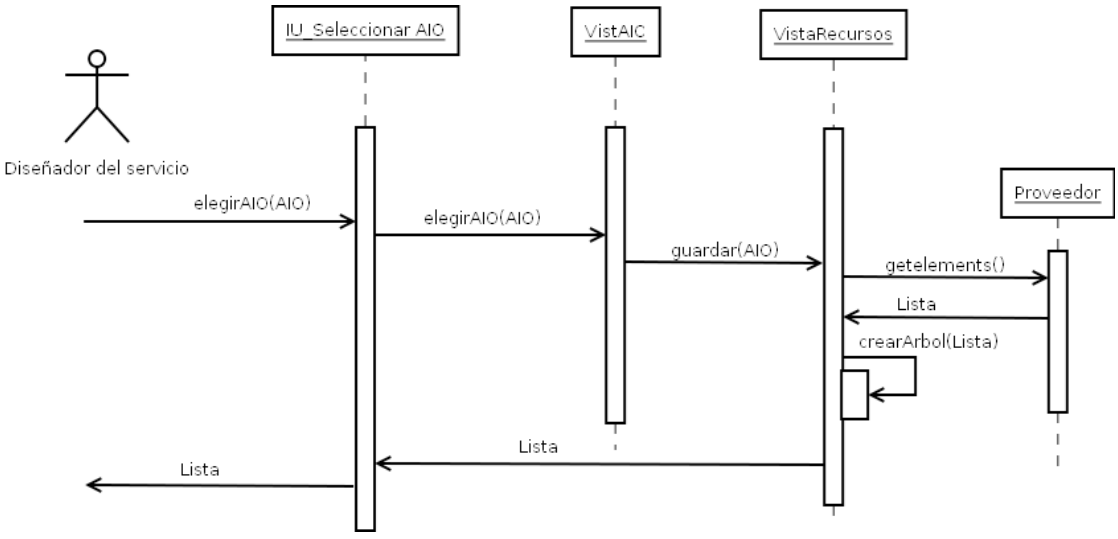

Figura 16: Diagrama de secuencia Seleccionar AIO

4.3.3.2 Seleccionar Recursos

El diseñador del servicio selecciona los recursos asociados a un componente de la interfaz gráfica. Los recursos seleccionados se asocian al AIO que se ha elegido anteriormente y esa asociación se guarda hasta que finalice el proceso de asociación (figura 17).

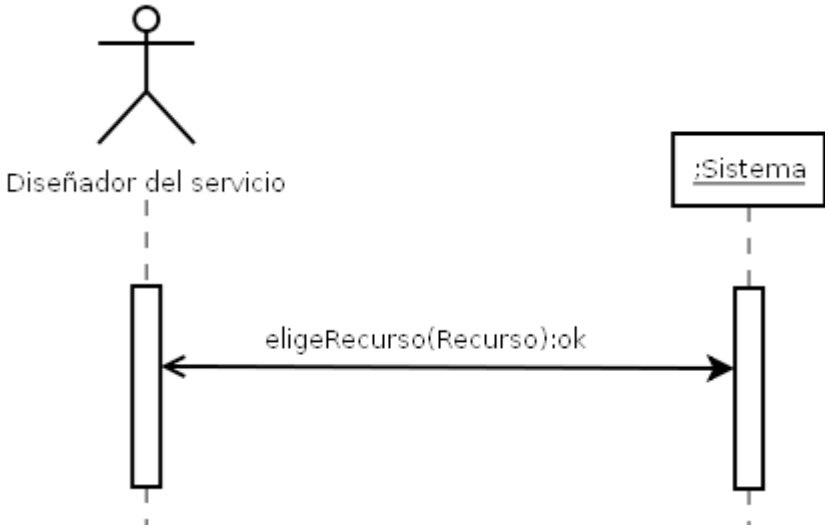

Figura 17: Diagrama de secuencia del sistema. Caso de uso "Seleccionar Recursos"

Los métodos planificados para este caso de uso son "guardaRecurso" y "asociarAioRecurso".

El método "guardaRecurso" de la clase VistaRecursos se encarga de guardar los recursos asociados al AIO que el diseñador del servicio selecciona del árbol mostrado. El método "asociarAioRecurso" crea una estructura de datos con el AIO tratado y los recursos seleccionados. El caso de uso correspondiente se representa en la figura 18.

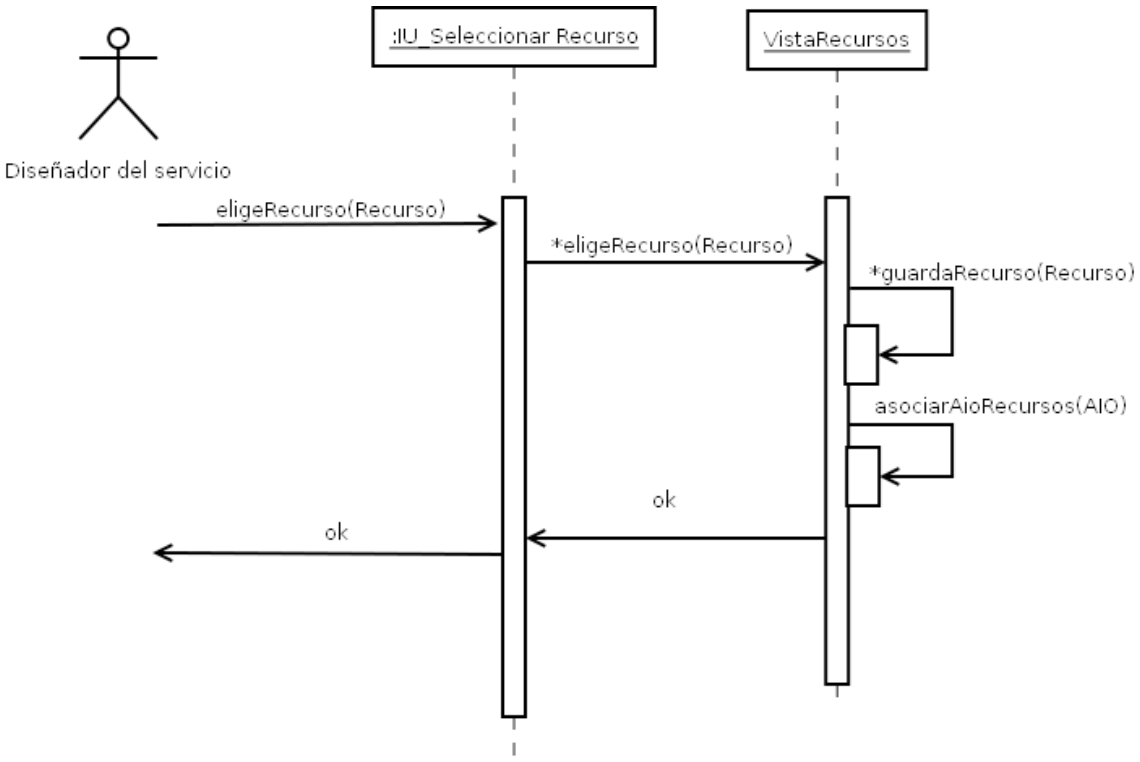

Figura 18: Diagrama de secuencia Seleccionar Recursos

4.3.3.3 Cancelar Asociación

El diseñador rechaza los recursos seleccionados, cancela la selección de los recursos asociados al AIO. Al realizar dicha acción el sistema muestra la tabla que contiene los componentes de la interfaz gráfica (figura 19).

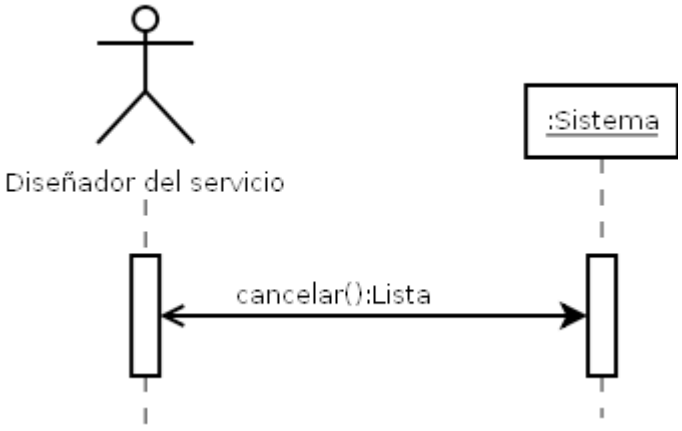

Figura 19: Diagrama de secuencia del sistema. Caso de uso "Cancelar asociación"

En la figura 20 se contemplan los métodos planificados para este caso de uso "anularAsociación", "abrir", "getElements" y "crearTabla".

El método anularAsociación de la clase VistaRecursos elimina la estructura de datos creada con el AIO tratado y sus recursos. El método abrir de la clase VistAIO lanza la vista que permite la selección de un AIO y cierra la vista actual que permite seleccionar los recursos. El método getElements de la clase ProveedorAIO agrupa y envía los AIO que se muestran en la tabla. El método crearTabla de la clase VistAIO crea una tabla donde se muestran los elementos de la lista de objetos AIO que le ha enviado el proveedor.

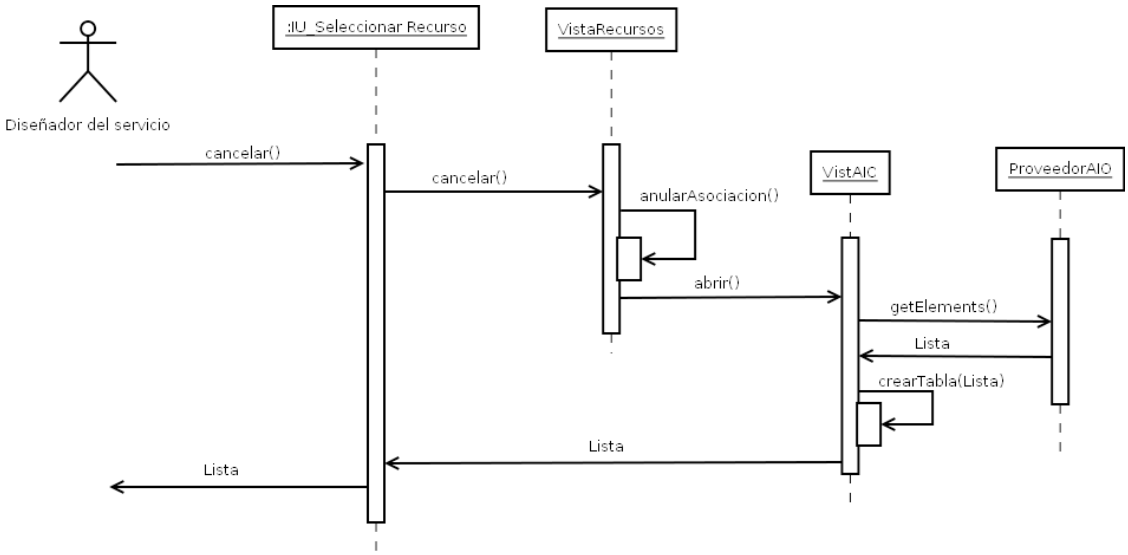

Figura 20: Diagrama de secuencia Cancelar Asociación

4.3.3.4 Guardar Asociación

El diseñador del servicio finaliza la selección de los recursos asociados al AIO y se guarda la asociación. El sistema recoge en la estructura de salida de datos, los recursos asociados a un componente de la interfaz gráfica. Los recursos seleccionados se asocian al AIO que se ha elegido anteriormente y esa asociación se guarda en la estructura de salida de datos (figura 21).

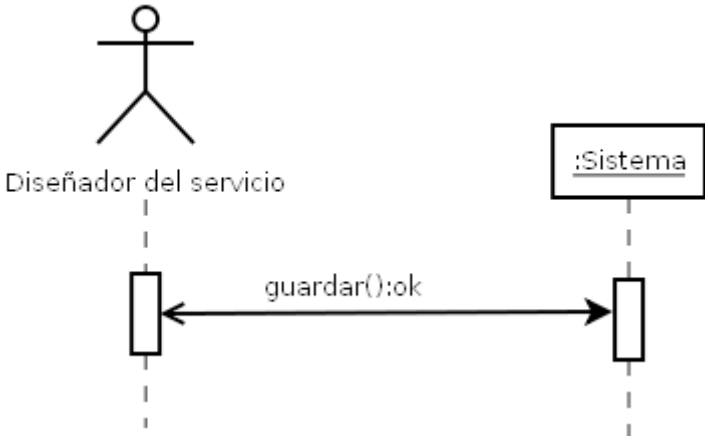

Figura 21: Diagrama de secuencia del sistema. Caso de uso "Guardar Asociación"

Como se puede observar en la figura 22, en este caso de uso se ha identificado solamente un método a desarrollar, el método "guardar". Este método de la clase ControlSalida guarda en la estructura de salida de datos, el AIO tratado y sus recursos asociados.

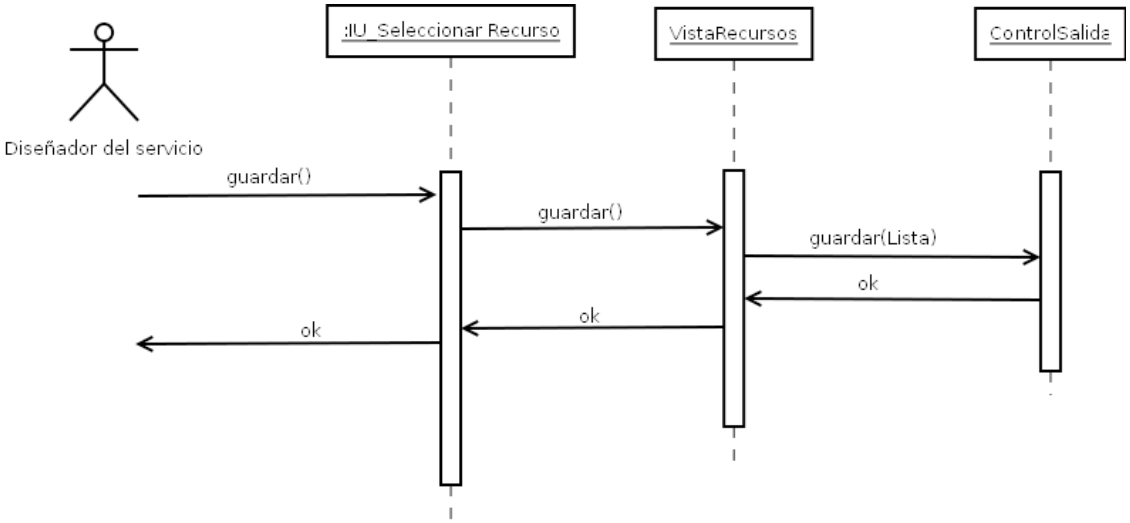

Figura 22: Diagrama de secuencia Guardar Asociación

4.3.3.5 Finalizar Asociación

El diseñador del servicio finaliza la asociación entre los AIO y sus respectivos recursos. Los recursos seleccionados se asocian al AIO que se ha elegido anteriormente y esa asociación se guarda en la estructura de salida de datos. La información guardada en la estructura de salida se añade al documento UIML, donde se describe la interfaz abstracta de usuario y los recursos de sus componentes (figura 23).

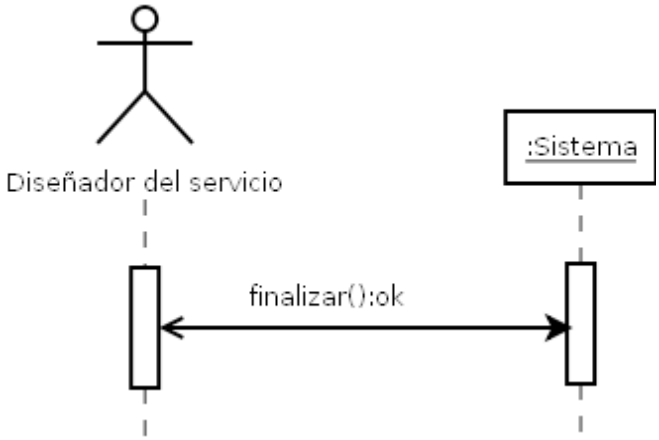

Figura 23: Diagrama de secuencia del sistema. Caso de uso "Finalizar Asociación"

En este caso de uso se han identificado el método "finalizar" para cerrar las vistas y el método "guardar" que cierra la aplicación. El método "finalizar" de la clase VistAIO cierra la vista que muestra la tabla de los AIO. El método "finalizar" de la clase VistaRecursos cierra la vista que muestra el árbol con los recursos disponibles. El método "guardar" de la clase ControlSalida guarda la estructura creada con los AIO tratados y sus recursos asociados. Esta estructura se trata y se añade al documento UIML que refleja la composición de la interfaz abstracta de usuario que el diseñador del servicio ha creado. La figura 24 muestra los métodos identificados.

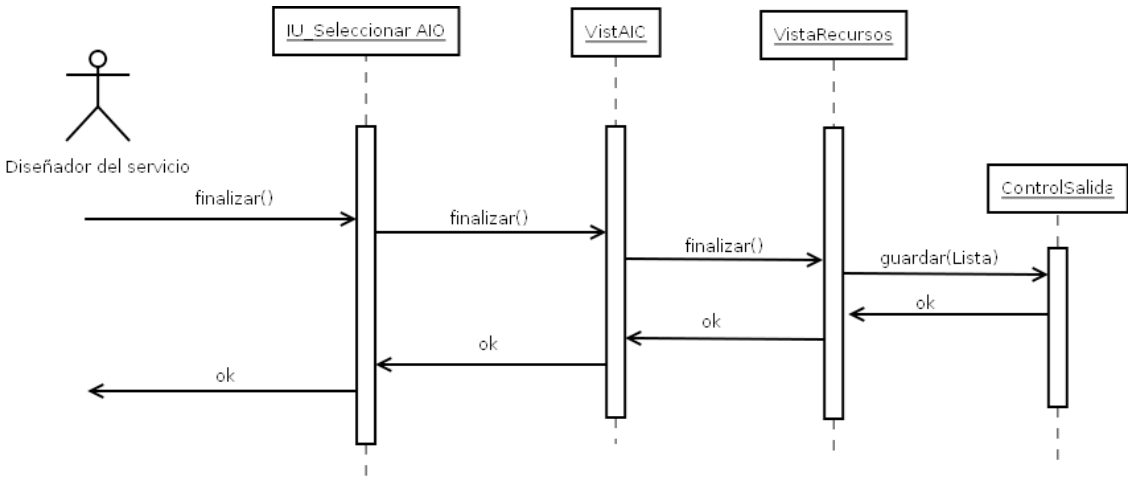

Figura 24: Diagrama de secuencia Finalizar Asociación

4.4 Interfaz Gráfica

En este apartado se muestran las interfaces de usuario creadas para la herramienta y se realiza una descripción de las mismas.

4.4.1 Interfaz del Editor Gráfico

Mediante esta interfaz, el diseñador del servicio ubicuo dispone de las herramientas necesarias para remodelar la estructura de la interfaz abstracta, que el sistema presenta.

La interfaz presentada por el sistema refleja la estructura que se desprende de la información recogida en el documento de entrada y que el sistema ha estandarizado a un documento UIML básico.

Partiendo de la estructura de interfaz abstracta presentada por el sistema, el diseñador es capaz de reorganizarla si lo considera oportuno, mediante el uso de las herramientas proporcionadas en la paleta del editor. Estas herramientas se han presentado en el apartado 4.3.2, también tiene la oportunidad de aceptar la estructura inicialmente presentada si considera que esa distribución es la correcta.

En todo caso, tras utilizar el editor gráfico el diseñador del servicio ubicuo logra que la estructura de la interfaz abstracta de usuario y la distribución de los objetos abstractos de interacción se vean reflejadas en el documento UIML, ampliando la información recogida en el documento UIML inicial presentado por el sistema.

La figura 25 ofrece una visión de la interfaz del editor gráfico, que se presenta al diseñador del servicio, mostrándole la estructura que se desprende del documento UIML de entrada.

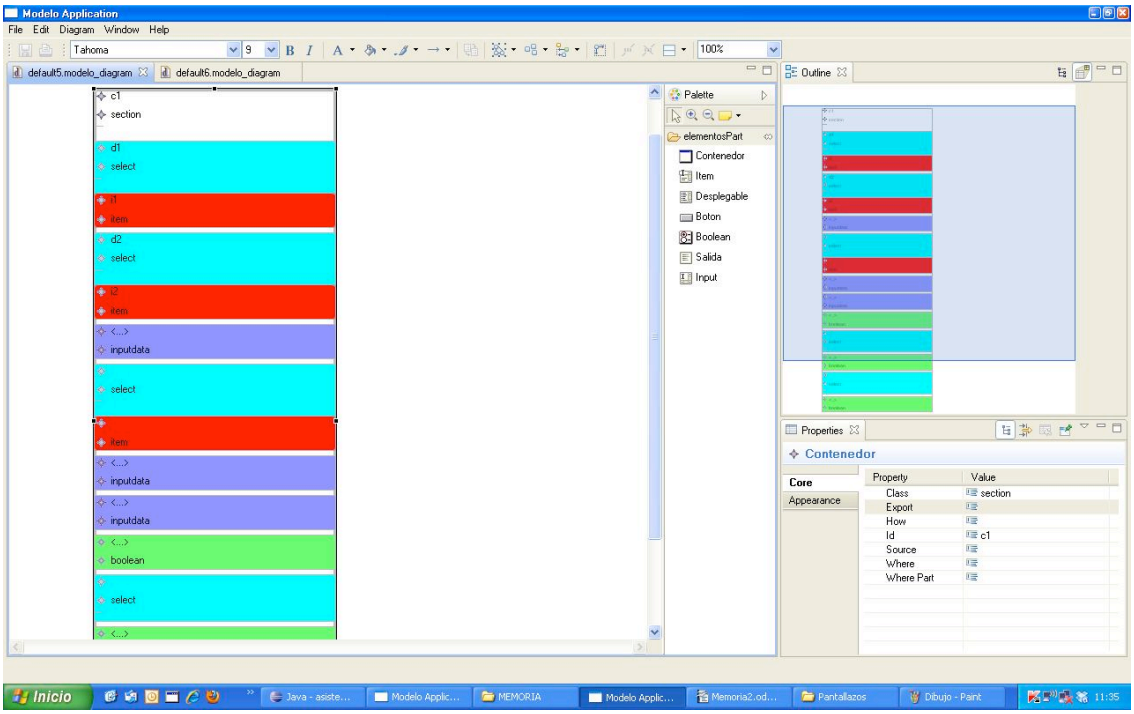

Figura 25: Editor gráfico

4.4.2 Interfaz de Selección de los AIO

Esta interfaz pertenece al módulo de asociación, que completa la recopilación de información para el documento UIML. Inicialmente se muestra la estructura de la interfaz abstracta de usuario en esta vista y los recursos asociados a los objetos abstractos de interacción se mostrarán en una vista asociada.

Como se ve en la figura 26, el diseñador del servicio ubicuo puede seleccionar los AIO que le interesen, para acceder a los recursos disponibles. Desde esta interfaz, también se controla el proceso de asociación, ya que contiene el botón que finaliza dicho proceso.

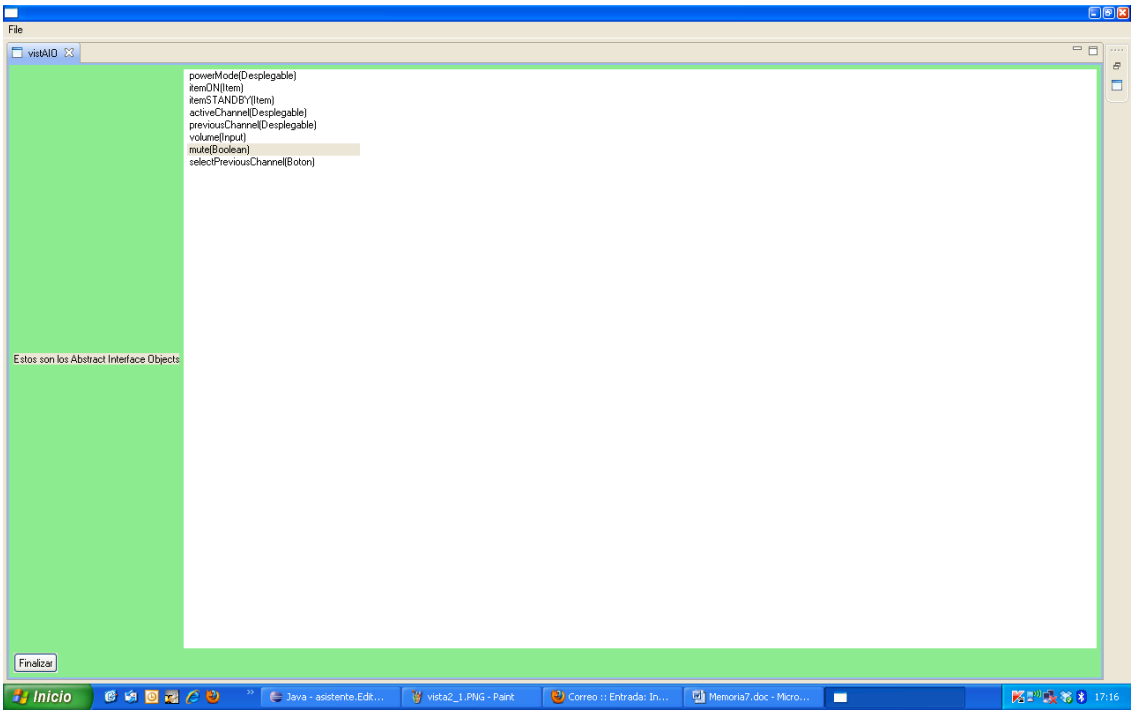

Figura 26: Interfaz que presenta los AIO

4.4.3 Interfaz de Recursos

Esta vista, perteneciente al modulo de asociación, es la que permite seleccionar los recursos multimedia a asociar. Tras seleccionar uno de los AIO mostrados en la anterior vista, se le ofrece al diseñador una vista de árbol con todos los recursos multimedia disponibles categorizados según su tipo. El diseñador tendrá que seleccionar todos los recursos que desea asociar al AIO seleccionado.

Concretamente, esta vista dispone de un elemento que muestra el nombre del AIO seleccionado anteriormente, en una ventana de texto para conocer a que se le están asociando los recursos. Una lista con los recursos disponibles desplegados en forma de árbol y agrupados según el tipo de recurso que es. Para la selección de recursos en el árbol, dispone de un checkbox por cada recurso, permitiendo la selección de varios de ellos. El diseñador puede seleccionar los recursos que considere necesarios y aceptar dicha selección o volver a la anterior interfaz para comenzar un nuevo proceso de asociación con otro objeto abstracto de interacción.

En esta interfaz el diseñador del servicio ubicuo realiza las asociaciones entre los AIO y sus recursos pero el proceso de asociación realmente termina, como ya se ha indicado, mediante la vista anterior que es la que permite salir de la aplicación.

La interfaz que se presenta al diseñador del servicio para asociar los recursos se muestra en la figura 27.

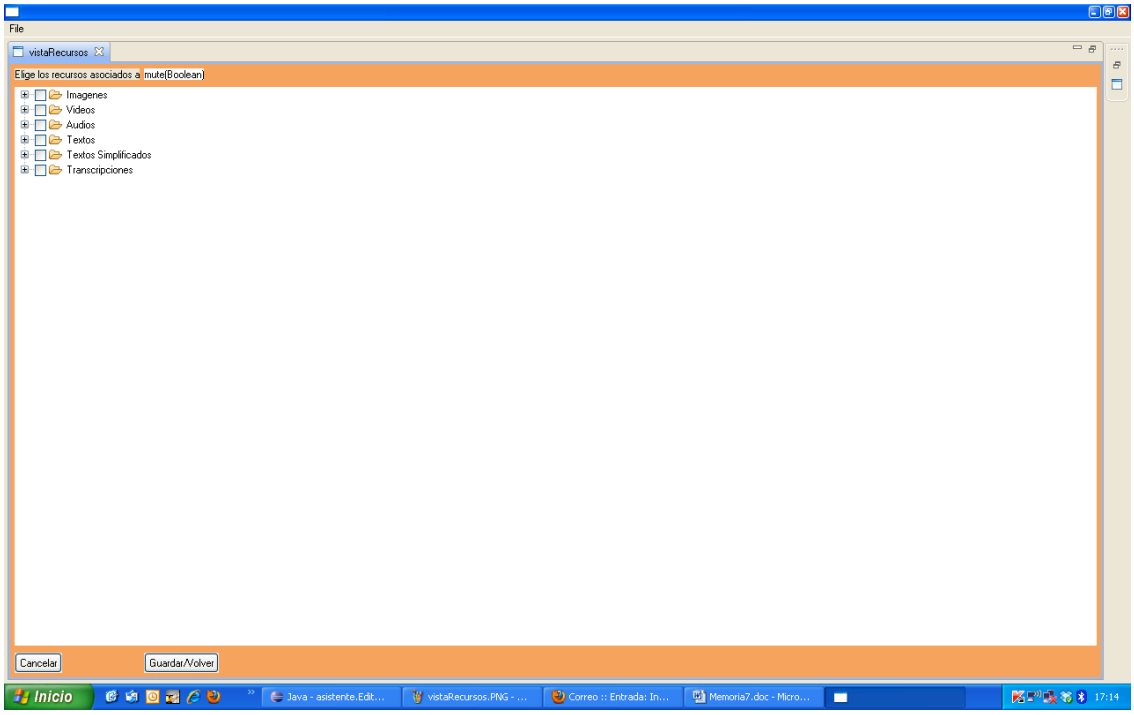

Figura 27: Interfaz que presenta los recursos disponibles

4.5 Implementación

En este apartado se muestran los detalles del desarrollo de la aplicación. El capítulo está dividido en dos partes referidas al editor gráfico y al módulo de asociación.

4.5.1 Editor Gráfico

En el apartado 4.2.1.1 y el apartado 4.2.1.2 se presentan los proyectos Eclipse EMF y GMF, que permiten crear modelos a partir de una sintaxis y editores gráficos a partir de un modelo respectivamente. Los detalles generales del mismo se han presentado en dichos apartados y aquí se mostrarán los distintos pasos realizados para crear el editor gráfico.

Como se puede ver en el apartado que se cita, el desarrollo del editor se compone de la creación y desarrollo de una serie de ficheros que van generando el aspecto del editor y el de su paleta de herramientas. A continuación se muestra ese desarrollo, para la creación de nuestro editor gráfico.

4.5.1.1 Modelo de Dominio (.ecore)

En el apartado 4.2.3 se muestra la idea que se ha seguido para modelar el documento UIML según nuestro interés. Se ha trabajado con el elemento “Part” que identifica a los AIO y se ha creado un modelo de dominio donde tomando dicho elemento como raíz, se

forma una estructura de herencia para modelar los distintos tipos de AIO que se han identificado como necesarios, para la paleta de herramientas del editor gráfico.

La descripción del modelo de dominio puede realizarse de manera gráfica o definiendo los componentes del dominio. El resultado final es el fichero ecore, con el que se genera el fichero genmodel. A su vez, el fichero genmodel es usado para generar el código java que describe nuestro modelo.

En el apartado 4.2.3 se puede ver la estructura pensada para la herencia del elemento "Part". Los hijos del elemento "Part" se agrupan en dos bloques dependiendo de si son elementos contenedores o son elementos finales de interacción. Estos a su vez se agrupan en otros dos bloques, dependiendo de si son elementos de entrada o elementos de salida.

El bloque de elementos contenedores contiene al objeto abstracto de interacción, que representa los elementos que pueden contener a otros elementos. En el bloque de elementos finales de interacción y en su rama de elementos de salida se ha definido el objeto abstracto de interacción que representa los elementos que muestran información, pero no se puede interactuar con los mismos. El resto de elementos finales de interacción son cinco elementos de entrada o disparadores: entrada de texto, booleano, desplegable, ítem de desplegable o botón.

En la figura 28 se ilustra la composición del fichero ecore creado donde se aprecian los distintos elementos. Éste modelo de dominio creado es utilizado como base para desarrollar a partir del mismo, el editor gráfico que se ha implementado.

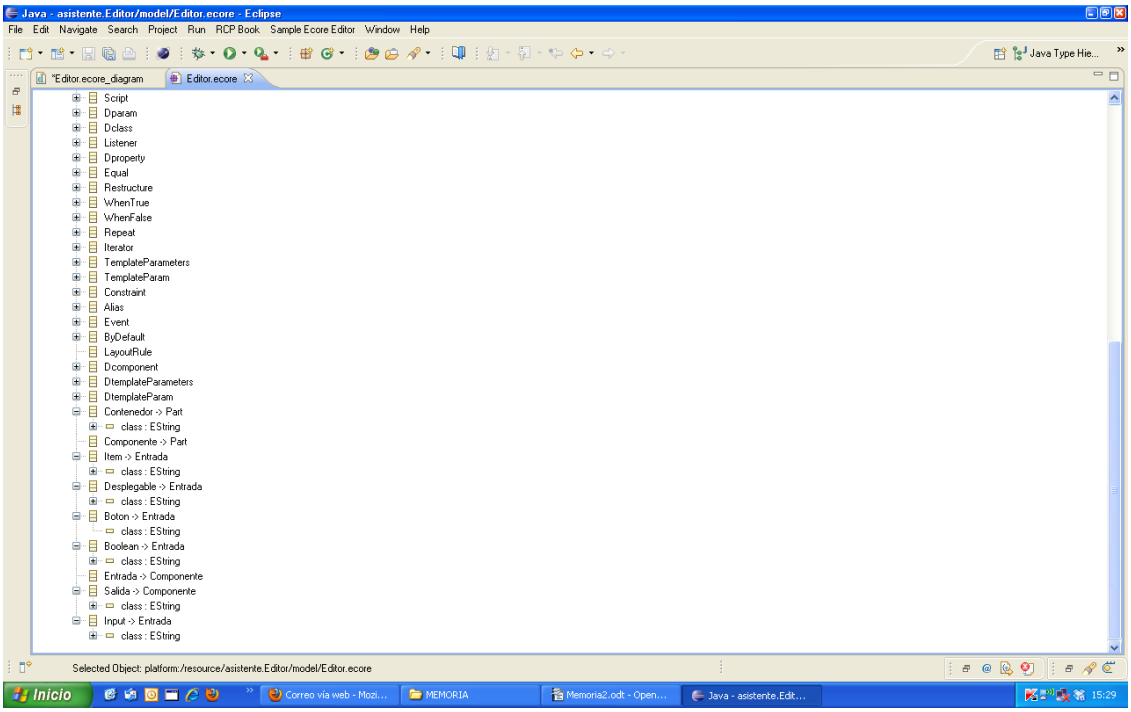

Figura 28: Fichero .ecore

4.5.1.2 Modelo Gráfico (.gmfgraph)

Como se observa en la figura 29, en este modelo se definen las características gráficas de los elementos identificados en el modelo de dominio, que aparecerán en el lienzo del editor. Los distintos elementos a utilizar se definen como tipo nodo. En nuestro caso no es necesario definir elementos de tipo enlace, los cuales permiten definir las conexiones de asociación y el flujo de secuencia, porque no se requieren para nuestro editor. Además, también se han definido elementos del tipo "compartimentos" para permitir el plegado y desplegado de los elementos que contienen a otros elementos.

Para comenzar la definición gráfica de los elementos, inicialmente se definen los nodos que se visualizarán en el lienzo del editor, que en nuestro caso son los elementos descritos en el anterior apartado. La definición de nuestros nodos comienza indicando la forma que tomarán y las etiqueta que mostrarán en el editor. Esta definición se realiza creando un elemento de tipo "Figure Descriptor" por cada nodo que queremos definir, desde el elemento "Figure Gallery" que indica el tipo de figura que usarán los nodos. Tras describir el formato que tendrán estos elementos en el editor, se define por cada uno de ellos un elemento tipo "Node" que hace referencia a las figuras creadas para dicho elementos. Como en la definición anterior, esta acción se realiza creando un hijo del tipo indicado por cada elemento. Posteriormente para acceder a las etiquetas definidas en "Figure Descriptor", se crean tantos elementos "Diagram Label" como etiquetas tenga el elemento. Para finalizar se indica cuales son las figuras que pueden desplegarse para contener a otros elementos, definiendo elementos de tipo "Compartment" para los elementos que son contenedores, en nuestro caso las figuras "Contenedor" y "Desplegable".

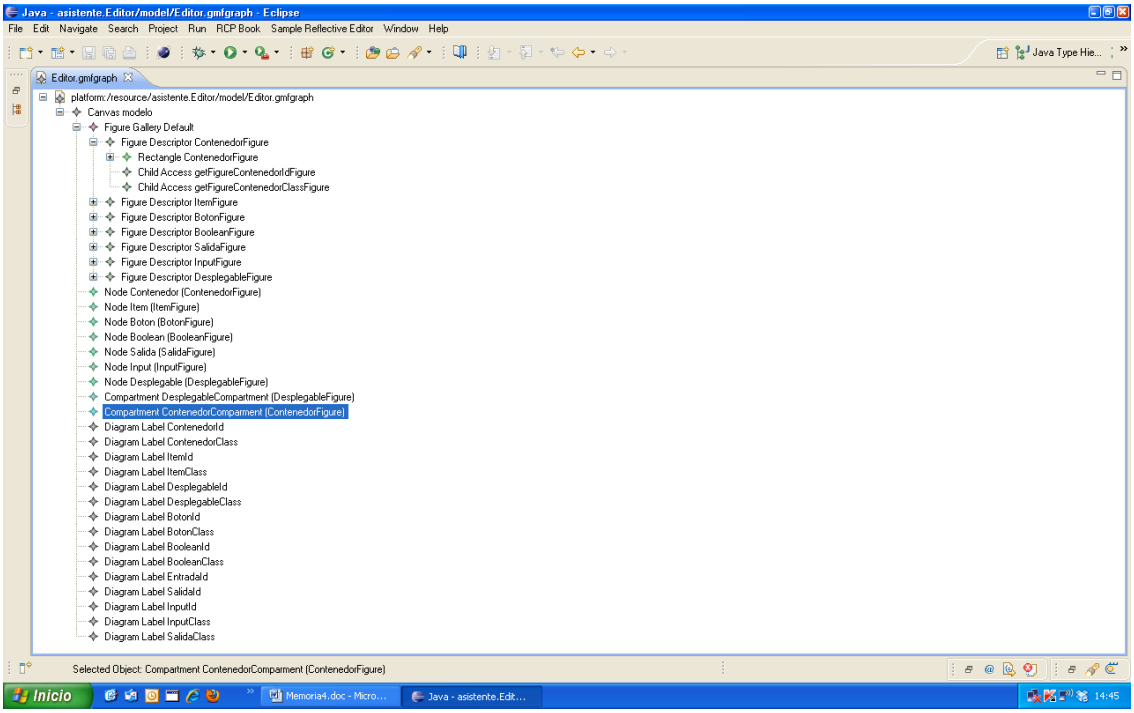

Figura 29: Fichero .gmfgraph

4.5.1.3 Modelo de Herramientas (.gmftool)

En este modelo, que se puede contemplar en la figura 30, se define la paleta de herramientas del editor. En nuestro modelo existe un grupo de herramientas ("Tool Group") llamado "elementosPart" que contiene las herramientas de creación ("Creation Tool") . Las herramientas sirven para crear un elemento en la paleta de herramientas de los definidos en el modelo gráfico.

Estas herramientas representan los distintos elementos "Part" representados en la interfaz abstracta. Como se puede ver en la figura 30, a cada herramienta de creación se le asigna una imagen para que al seleccionar un elemento de la paleta esa imagen indique al diseñador que tipo de figura está utilizando para definir la interfaz abstracta de usuario.

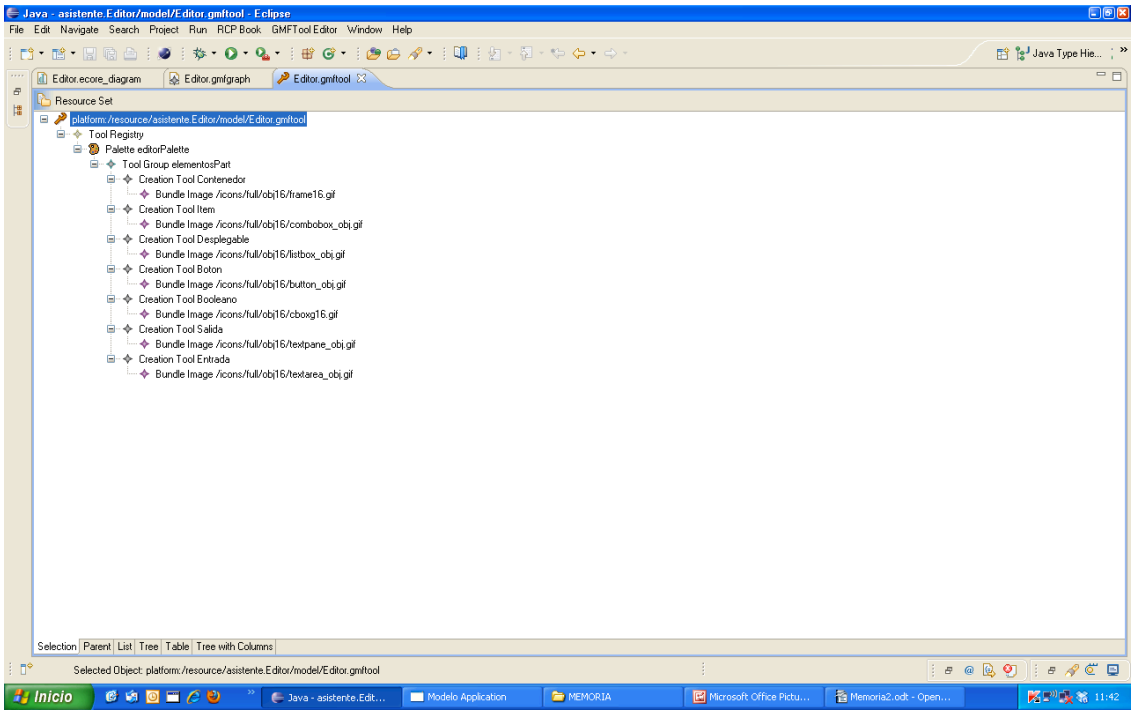

Figura 30: Fichero .gmftool

4.5.1.4 Modelo de Mapeo (.gmfmap)

En este punto del proceso se tienen tres modelos: del dominio, de definición gráfica y de definición de la paleta de herramientas. Para unir todos los modelos y definir las relaciones entre elementos de los distintos modelos es necesario crear un modelo que una todos los conceptos representados en los tres anteriores. Este es el modelo de mapeo. El modelo de mapeo utiliza los elementos del modelo del dominio, asignándoles una figura del modelo gráfico y el elemento del modelo de la definición gráfica de la paleta con la que se dibujarán.

Hay que generar los elementos que queremos dibujar en el lienzo como nodos. El nodo que sirve de marco a la mayoría de nodos es el elemento “Contenedor”. Los nodos que se pueden dibujar dentro del anterior se definen como elementos del tipo “Child Reference” mediante la opción “Properties” de “Top Node Reference”. El atributo “Child” de “Properties” tomará el nombre del elemento que queremos dibujar, y dicho elemento será definido dentro de “Child Reference” como un nodo.

Para ello es necesario definir un elemento del tipo “Node Mapping” mediante la opción “Properties” y se le asigna el elemento al que hace referencia en el modelo ecore, la definición de dicho elemento en el modelo .gmfgraph y en el modelo .gmftool.

Para incluir las etiquetas definidas en el modelo .gmfgraph se necesita definir un elemento del tipo “Feature Label Mapping” dentro del nodo generado.

La definición de la figura como desplegable cuando tiene elementos dentro se define mediante un elemento de tipo “Compartment Mapping” que hace referencia a dicho contenedor.

Hasta aquí se ha explicado el proceso de definición de un nodo contenedor, en nuestra paleta son el elemento “Contenedor” que contiene a los demás elementos menos “Botón” y “Salida” y el elemento “Desplegable” que contiene al elemento “Item”, el resto de nodos que no son contenedores deben ser definidos del tipo “Node Mapping” mediante la opción “New Child” de “Top Node Reference”, definiendo también para ellos las etiquetas del modelo .gmfgraph. En la figura 31 se muestra el modelo de mapeo desarrollado.

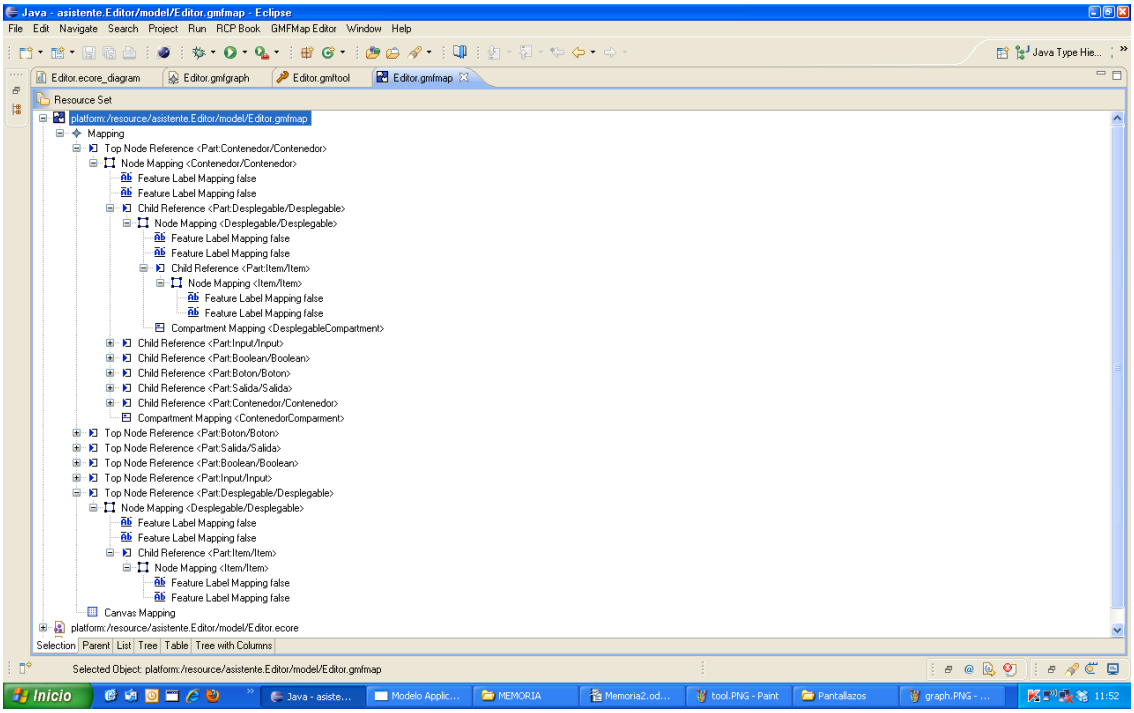

Figura 31: Fichero .gmfmap

4.5.1.5 Modelo Generador (.gmfgen)

El modelo generador se crea automáticamente a partir del modelo de mapeo y del fichero gmfgenmodel. Relaciona estos dos modelos como último paso antes de la generación del código del editor. La figura 32 ilustra el modelo generador creado.

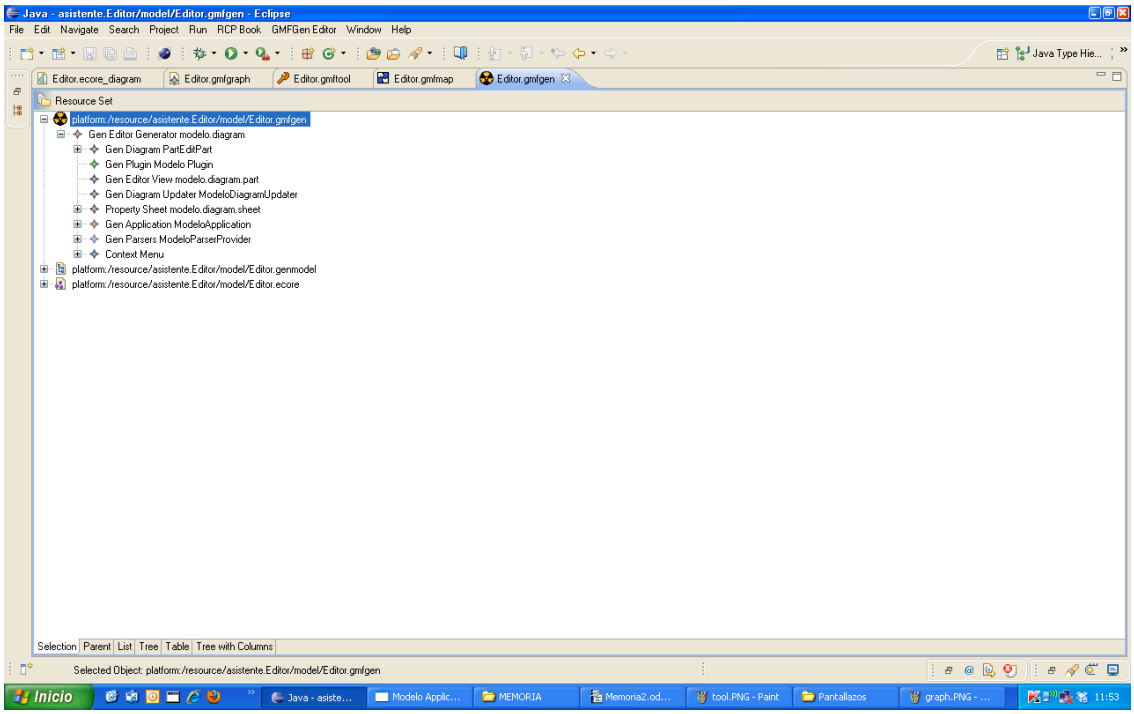

Figura 32: Fichero .gmfgen

4.5.1.6 Carpeta .diagram

Tras definir esta serie de modelos, mediante el último modelo creado, el modelo generador, se crea el editor propiamente dicho utilizando el botón derecho del ratón sobre este fichero y seleccionando la opción “Generate Diagram Code”. Haciendo esta operación se crea el editor que se ha definido y ya lo tenemos listo para usarlo.

En la figura 33 se puede observar el editor creado para la creación de interfaces abstractas de usuario.

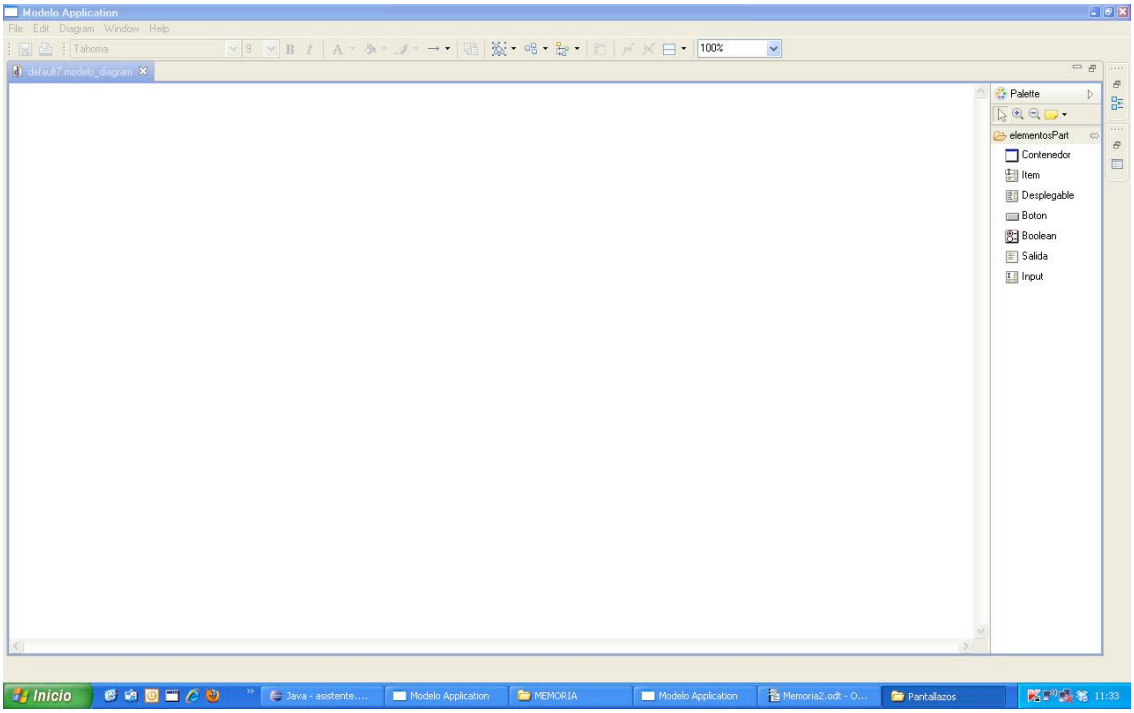

Figura 33: Editor gráfico

Este editor gráfico muestra en su paleta de herramientas las distintas opciones que tiene el diseñador del servicio para poder crear descripciones abstractas de interfaces de usuario de una forma más rápida e intuitiva. El diseñador del servicio seleccionando las distintas herramientas de las que dispone puede crear de forma gráfica una composición de elementos abstractos de interacción que refleje las relaciones entre los mismos y una vez termine la composición se genera un documento UIML donde se pueden ver esas relaciones.

4.5.2 Módulo de Asociación

La implementación del módulo de asociación se compone de varios paquetes, algunos creados automáticamente por los proyectos de Eclipse utilizados en la creación de la herramienta gráfica y otros creados expresamente para el módulo. Se han utilizado cuatro paquetes principales, que corresponden a las capas de la aplicación: presentación, modelo de negocio y datos.

Dichos paquetes son:

- asistente.asociar
- asistente.asociar.vistas
- asistente.asociar.proveedor
- asistente.asociar.datosEntrada

La presencia de cuatro paquetes y tres capas de la aplicación indica que algún paquete contiene clases que implementan métodos correspondientes a distintas capas de la aplicación. Esto es así porque la implementación comienza con unos paquetes ya creados automáticamente y hay que adaptar sus clases y métodos a las necesidades de la herramienta gráfica, además de crear los paquetes y clases necesarios para la correcta implementación de la misma.

En la capa de presentación se implementan métodos de las clases VistAIO y VistaRecursos del paquete asistente.asociar.vistas creado específicamente para la herramienta gráfica. También se implementan métodos de la clase Perspective del paquete asistente.asociar, este paquete se crea automáticamente al desarrollar un proyecto RCP.

La capa de negocio se corresponde con la implementación de métodos que pertenecen a las clases VistAIO y VistaRecursos del paquete asistente.asociar.vistas.

El paquete asistente.asociar.proveedor contiene las clases que proveen de datos a las vistas, estas clases son VistAIOContentProvider, VistAIOLabelProvider, VistaRecursosContentProvider y VistaRecursosLabelProvider.

Las clases VistAIOContentProvider y VistaRecursosContentProvider anteriormente mencionadas se apoyan en las clases Resources y ResourcesAssociation del paquete asistente.asociar.datosEntrada para proveer los datos a las estructuras creadas en las vistas que componen este módulo de asociación.

Observando las clases principales que se han implementado y contemplando los paquetes donde se alojan, se descubre que la distribución de los paquetes entre las capas de la aplicación queda de forma que en la capa de presentación se utilizan clases de los paquetes asistente.asociar y asistente.asociar.vistas, en la capa de negocio se emplean clases del paquete asistente.asociar.vistas y en la capa de datos clases del paquete asistente.asociar.datosEntrada, y del paquete asistente.asociar.proveedor. De los cuatro paquetes principales que se han utilizado, el paquete asistente.asociar.vistas se utiliza en dos de las capas de la aplicación y el resto de paquetes principales se utilizan específicamente en una de ellas.

En la figura 34 se puede observar un diagrama de clases que muestra las relaciones entre las clases utilizadas en la creación de la herramienta gráfica.

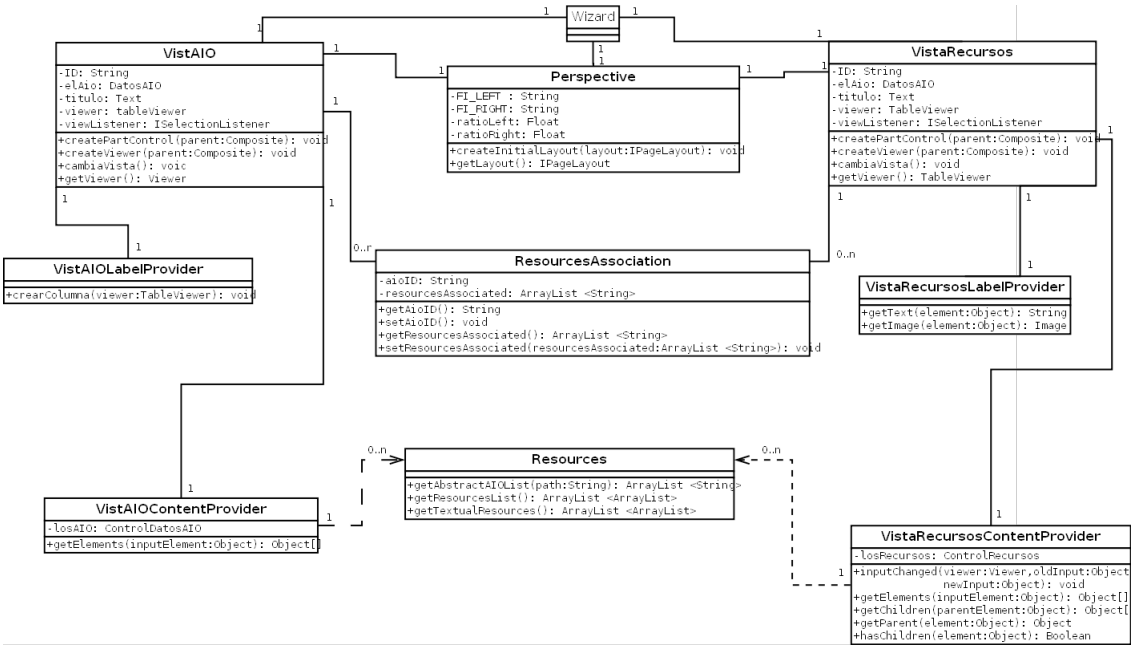

Figura 34: Diagrama de clases del módulo de asociación

4.5.2.1 Capa de Presentación

Partiendo de la capa de presentación se inicia la ventana que muestra la tabla con los AIO que forman parte de la interfaz abstracta de usuario definida en el editor gráfico. Para presentar la primera ventana, se ajusta la implementación de la clase Perspective del paquete asistente.asociar. Este paquete se crea automáticamente al crear una aplicación Eclipse RCP [16] y en la clase Perspective se describe la disposición inicial de las vistas y editores que la aplicación mostrará.

Se ha decidido crear una aplicación que muestre una vista, donde se vea una tabla con los AIO que se han utilizado en la descripción abstracta de la interfaz de usuario, del editor gráfico. Esta vista se llama VistAIO. Al seleccionar un AIO de la tabla mostrada, se lanza otra vista que muestra los recursos disponibles para el AIO seleccionado en la vista anterior, a esa vista se la denomina VistaRecursos.

En la clase Perspective se han definido las dos vistas citadas anteriormente para que al comienzo de la aplicación se muestre VistAIO, mientras VistaRecursos queda oculta hasta que se seleccione un AIO de la tabla de VistAIO.

En las clases VistAIO y VistaRecursos se definen las composiciones de las dos vistas en el método “createPartControl” que ambas implementan. Este método se utiliza para crear los componentes de la vista y la distribución de éstos.

Eclipse RCP utiliza las librerías gráficas SWT y JFace para realizar interfaces de usuario. SWT es un conjunto de componentes que sirve para construir interfaces gráficas en Java. Por su parte JFace es una capa de abstracción sobre lo que es SWT, lo que nos brinda una amplia gama de componentes reusables facilitándonos la creación de

interfaces gráficas. Utilizando las librerías gráficas SWT y JFace se resuelve uno de los principales problemas que se presentaban al utilizar la librería gráfica Swing. Utilizando Swing en vez de utilizar los objetos predefinidos del sistema operativo (botones, ventanas, etc) se crean los propios, lo que lleva a un mayor procesamiento, la UI desarrollada en Swing, se genera igual independientemente del Sistema Operativo. Con SWT y JFace en vez de crear sus propios controles, se utilizan los predefinidos del Sistema Operativo, haciendo las aplicaciones desarrolladas con estas librerías mucho más rápidas y con el look and feel particular de cada Sistema Operativo.

4.5.2.2 Capa de Negocio

La capa de negocio se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados, y con la capa de datos para solicitar, almacenar o recuperar datos de él. En la implementación del editor de asociación este patrón se utiliza en las clases correspondientes con las VistAIO y VistaRecursos ya que por la estructura de Eclipse RCP y su uso de JFace, las vistas implementan mecanismos de acción y escucha de esas acciones.

En el editor de asociación la acción más importante es la selección de un AIO en la tabla, que se muestra en VistAIO, y su posterior aparición como cabecera de la vista, que muestra los recursos que se pueden asociar al mismo. La acción se implementa en la VistAIO en su método “createViewer”, este método crea la tabla que se muestra en la interfaz y maneja todos los datos que aparecen en el mismo.

La escucha de dicha acción se realiza en la clase VistaRecursos que es la encargada de recoger el objeto enviado y tratarlo según sea necesario, en este caso recoge la selección y presenta el identificador del objeto como título de la interfaz que se presenta, tras seleccionar un AIO en la tabla de presentación de los AIO de la interfaz abstracta de usuario.

4.5.2.3 Capa de Datos

La capa de datos es donde residen los datos y es la encargada de acceder a los mismos. Nuestra capa de datos está dividida entre datos de entrada para los AIO estructurados en el editor gráfico y para los recursos, y los datos de salida que son el resultado de las asociaciones entre objetos AIO y sus recursos.

Los datos de entrada se cargan en la clase VistAIOContentProvider que se encarga de rellenar la tabla que muestra los AIO estructurados en el editor gráfico. La clase VistAIOContentProvider esta encargada de proveer los datos de la clase VistAIO. Al crear la tabla en esta clase se hace una llamada a su proveedor de contenido.

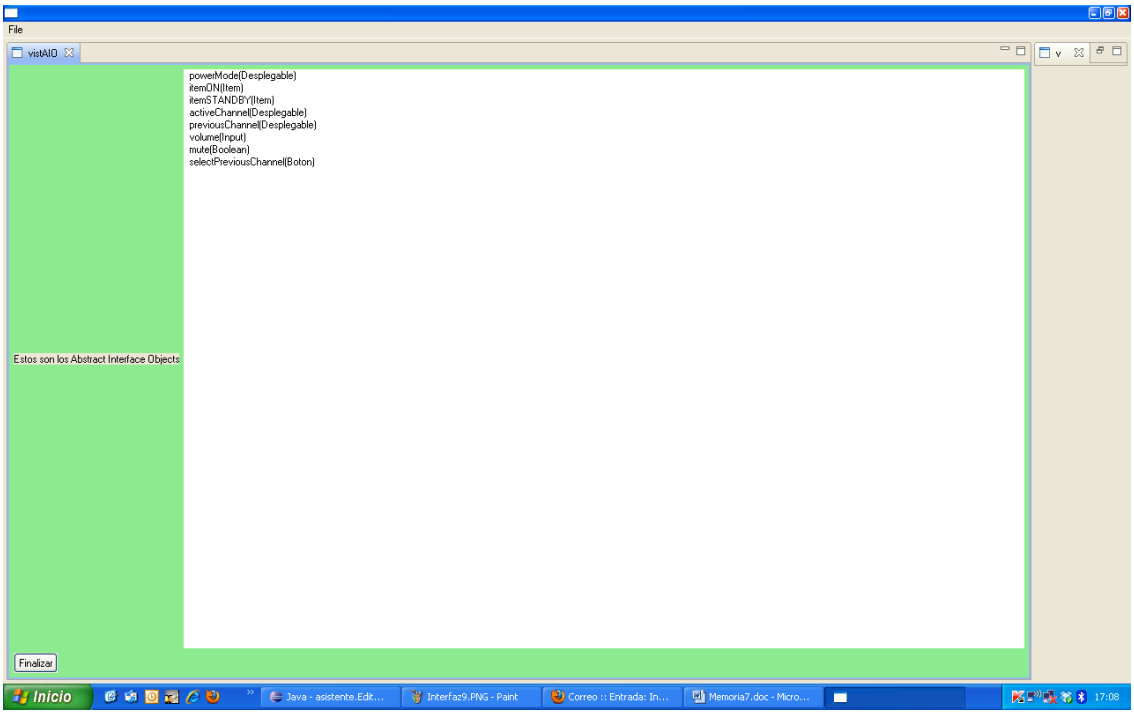

Figura 35: VistAIO

En la figura 35 se muestra la interfaz inicial que se le muestra al diseñador del servicio con la lista de los AIO que puede seleccionar, la vista con los recursos disponibles está oculta en este punto del proceso. Al seleccionar uno de los elementos de la tabla, se lanza la interfaz que muestra los recursos que se pueden asociar al AIO elegido (ver figura 36).

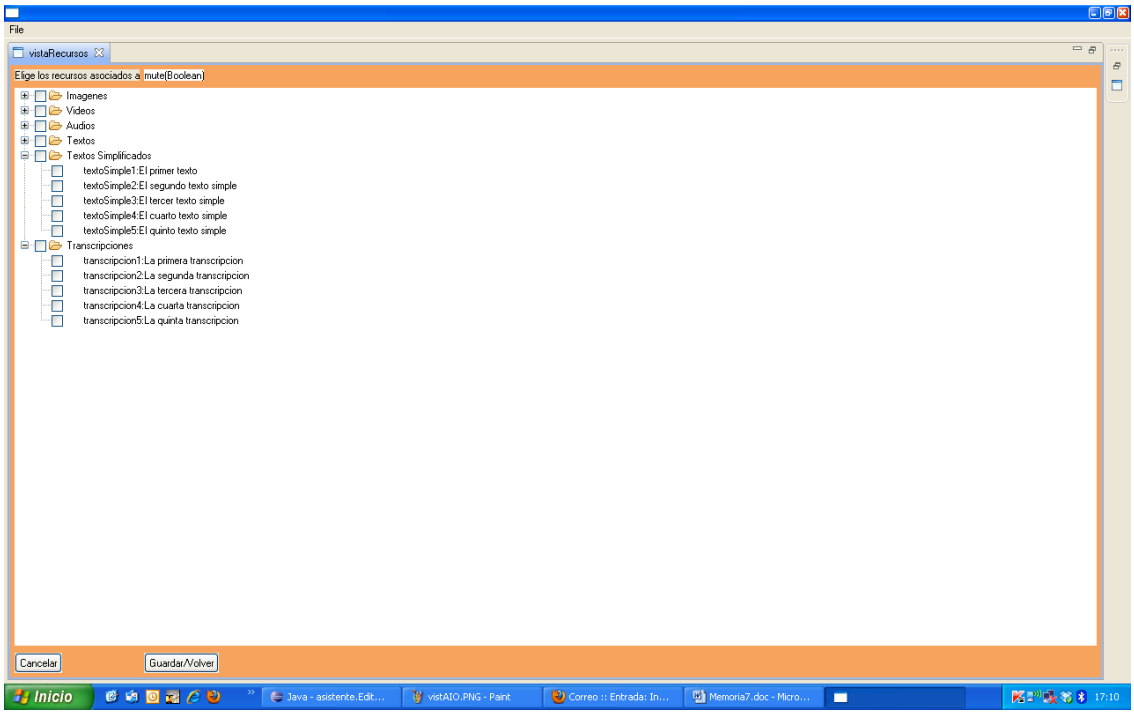

Figura 36: VistaRecursos

La clase VistaRecursos es la encargada de implementar la vista que muestra los recursos que se pueden asociar al AIO seleccionado. El identificador del AIO seleccionado aparece en el campo de texto que se sitúa en la parte superior del árbol que muestra los recursos. El manejo de esta situación se ha explicado en el apartado 4.5.2.2 .La clase VistaRecursos obtiene los datos que muestra, de su proveedor que es la clase Resources.

La clase VistaRecursos para crear el árbol de recursos llama a su proveedor de recursos en el método “createViewer”, de la misma manera que lo hace la clase VistAIO, vease la página 67.

Tras asociar los AIO con sus recursos y pulsando el botón “Guardar/Volver” que se puede ver en la figura 36, se actualiza el fichero UIML de salida con las asociaciones que el diseñador del servicio ha realizado.

CAPÍTULO 5 PRUEBAS

Este capítulo está dedicado a las pruebas realizadas a la herramienta gráfica creada, consta de un subapartado dedicado al diseño de las mismas y otro subapartado donde se muestran los resultados obtenidos.

5.1 Diseño de las pruebas

Para validar la implementación realizada se han diseñado una serie de pruebas funcionales sobre las funcionalidades descritas en los casos de uso.

Para validar la funcionalidad del editor gráfico se han diseñado las siguientes pruebas:

- Prueba 1: Seleccionar archivos de servicio
  Resultado esperado: Mostrar la interfaz abstracta de usuario que ofrece el sistema.

- Prueba 2: Adaptar interfaz
  Resultado esperado: Las herramientas de la paleta del editor permiten adaptar la interfaz presentada por el sistema al formato deseado por el diseñador del servicio ubicuo.

- Prueba 3: Elegir componente
  Resultado esperado: El elemento seleccionado en la paleta del editor es realmente el elemento que queremos elegir.

- Prueba 4: Reorganizar interfaz
  Resultado esperado: Mover los elementos gráficos sobre el lienzo del editor.

- Prueba 5: Quitar componente
  Resultado esperado: Borrar del lienzo del editor el elemento seleccionado.

- Prueba 6: Modificar componente
  Resultado esperado: Modificar el tamaño y la posición de un elemento del editor.

- Prueba 7: Añadir Componente
  Resultado esperado: Tener un nuevo componente en la interfaz.

- Prueba 8: Finalizar adaptación
  Resultado esperado: Documento UIML con las características de la interfaz abstracta de usuario que se ha creado.

En la siguiente lista se ofrecen las pruebas diseñadas para el módulo de asociación:

- Prueba 1: Seleccionar AIO
  Resultado esperado: Seleccionar un AIO de la lista y que su nombre aparezca en la vista de recursos que se abre tras la selección.

- Prueba 2: Seleccionar recursos
  Resultado esperado: Seleccionar los recursos asociados al AIO.

- Prueba 3: Guardar asociación
  Resultado esperado: Al pinchar en "Guardar/Volver" se guarda la estructura de datos creada con el AIO tratado y sus recursos.

- Prueba 4: Cancelar asociación
  Resultado esperado: Al pinchar en "Cancelar" se cierra la vista que presenta los recursos y se abre la vista que presenta los AIO.

- Prueba 5: Finalizar asociación
  Resultado esperado: Al pinchar en "Finalizar" se cierra la aplicación, se guardan las asociaciones creadas y se genera un documento UIML donde se reflejan las asociaciones entre los AIO y sus recursos.

5.2 Resultado de las pruebas

En este apartado se muestra el resultado de las pruebas funcionales diseñadas en el apartado 5.1.

Editor gráfico

- La primera prueba diseñada se refiere a la presentación de una interfaz abstracta de usuario después de que el diseñador del servicio ubicuo selecciona el documento XML donde se especifican las características del servicio. Tras realizar el proceso de selección del documento, se comprueba que la herramienta gráfica muestra la interfaz abstracta de usuario, donde se aprecia la distribución de los objetos abstractos de interacción que se definen en el documento.
- En la segunda prueba se adapta la interfaz presentada por el sistema a las necesidades del diseñador del servicio, para ello se comprueba que las herramientas que se ofrecen en la paleta del editor permiten crear en su forma y posición los objetos gráficos que representan a los objetos abstractos de interacción. Realizadas varias pruebas con cada elemento de la paleta del editor se verifica que la paleta de herramientas permite realizar las composiciones de interfaz que interesen al diseñador del servicio.
- La anterior prueba engloba a las pruebas tres, cuatro, cinco, seis y siete que se pueden observar en el anterior apartado. Se ha comprobado que el diseñador del servicio puede seleccionar los elementos de la paleta del editor, puede mover los componentes del lienzo del editor por toda su superficie, puede borrar y añadir

elementos al lienzo del editor y esos elementos añadidos pueden ser modificados en su tamaño y posición sobre el lienzo.

● Finalmente la última prueba diseñada para el editor gráfico, se refiere a la acción de finalizar el diseño de la interfaz gráfica y la creación del documento que muestra la composición creada en el editor para su posterior manejo en la vista que ofrece los AIO. Realizada la prueba se observa que la entrada de datos a la vista que muestra los objetos abstractos de interacción se realiza correctamente.

Módulo de asociación

Para el módulo de asociación se han diseñado una serie de pruebas relativas a las selecciones que realiza el diseñador entre los AIO y los recursos que se muestran, y al correcto funcionamiento de los botones que forman parte de las vistas.

● En la primera prueba diseñada se comprueba que el nombre del AIO seleccionado se muestra en la vista que ofrece los recursos disponibles, durante la realización de las pruebas se observa que si el nombre del AIO seleccionado es demasiado largo el campo de texto donde se muestra oculta parte del mismo, tras localizar el lugar del código donde se define el campo de texto se modifica su configuración y se confirma su correcto funcionamiento.

● La segunda prueba diseñada comprueba que el diseñador del servicio pueda realizar las selecciones que estime oportunas en el árbol de recursos que le ofrece la herramienta gráfica. Se comprueba que tras realizar varias selecciones en el árbol de recursos, si se pulsa en el botón “Guardar/Volver” las selecciones realizadas se mantienen durante todo el procedimiento de asociar recursos a los AIO en el caso de repetir la selección de un AIO. Este resultado se refiere a la tercera prueba diseñada.

● En la cuarta prueba diseñada se quiere comprobar el funcionamiento del botón “Cancelar”. Pulsando ese botón se cierra la vista que ofrece los recursos que se pueden asociar al AIO y se abre la vista donde pueden seleccionarse los AIO que forman la interfaz abstracta de usuario creada por el diseñador del servicio. La prueba se realiza correctamente.

● La última prueba examina el correcto cierre de la aplicación tras realizar las asociaciones que el diseñador estima oportunas y la generación del documento UIML, donde se indica la estructura de la interfaz abstracta de usuario y los recursos asociados a sus componentes.

Durante la realización de esta prueba se ha comprobado, que pulsando en el botón “Finalizar” se generaba el documento pero la aplicación quedaba en estado de pausa sin llegar a cerrarse. Este hecho informa de algún error en el código referido a ese elemento de la vista que muestra los AIO. Encontrado el fragmento de código donde se producía el error se codifica correctamente esa función del elemento “Finalizar” y el error queda solucionado.

CAPÍTULO 6 INTEGRACIÓN EN EGOKI

El alcance de este proyecto no incorpora la generación completa de un documento UIML, debido a que es un proyecto de fin de carrera y se excedían demasiado las horas para generar el documento UIML completo.

Concretamente este proyecto automatiza la generación de los elementos UIML *head, structure, style y content* que son los más densos para generar manualmente. Por el contrario, los elementos *behavior y peers* no se han tenido en cuenta. Por ello, para la integración en el sistema Egoki, del documento UIML generado a través de la herramienta se deben incorporar manualmente los elementos no considerados (behavior y peers).

Una vez se hayan integrado dichos elementos, se considera que el documento UIML esta completo. Tras ello se debe copiar este documento UIML al sistema Egoki y éste será capaz de generar una interfaz de usuario funcional y adaptada a las características del usuario para el servicio proporcionado.

CAPÍTULO 7 CONCLUSIONES Y TRABAJO FUTURO

Este capítulo muestra las conclusiones del autor sobre el desarrollo del proyecto y las posibilidades que se abren en el futuro para ampliar el proyecto creado.

7.1 Conclusiones

En este apartado se enumeran las conclusiones a las que he llegado tras comprobar el cumplimiento de los objetivos propuestos.

Se pretendía una herramienta de alto nivel (intuitiva, textual y gráfica) para poder enriquecer los servicios ubicuos de una forma sencilla e intuitiva, en lugar de tener que implementar código XML y mi conclusión, basándome en las pruebas realizadas, es que la herramienta creada cumple esos objetivos.

A nivel práctico se ha separado la creación de la descripción de la interfaz, de la asociación de recursos mediante la creación de un editor gráfico y un módulo de asociación, lo que simplifica la tarea al diseñador del servicio, dándole la oportunidad de crear la estructura de la interfaz olvidándose de los recursos de cada componente de la misma y una vez terminada esa tarea pueda centrarse en los recursos que cada componente de la interfaz tenga asociados.

El asistente creado genera automáticamente el documento UIML donde se muestra la composición de la interfaz abstracta de usuario y los recursos asociados a los componentes de esa interfaz, ahorrando al diseñador la tarea de crearlo o la necesidad de conocer el lenguaje XML.

El editor gráfico tiene una interfaz agradable y muy intuitiva, donde los componentes que se ofrecen en la paleta de herramientas informan gráficamente al diseñador del tipo de AIO que puede manejar, además es totalmente funcional para los objetivos que se buscan.

El módulo de asociación se presenta de una forma gráfica sencilla, pero muy funcional, esa sencillez del editor hace que su uso sea muy intuitivo y además cada componente del editor muestra un texto de información de sus funciones para ayudar al diseñador a comprenderlos, facilitando su uso.

Trabajar en este proyecto ha supuesto también un acercamiento a los entornos ubicuos y a la creación de tecnologías accesibles para personas con necesidades especiales, temas que me parecen muy interesantes y que deben recibir un impulso en su investigación y desarrollo.

En lo referido al proceso de desarrollo del proyecto, tengo que decir que se ha trabajado con unas herramientas que necesitan un período de aprendizaje bastante alto. A la hora

de aprender los conceptos necesarios sobre EMF, GMF y RCP, me he encontrado con que estas herramientas son de uso minoritario en nuestro entorno y esa condición hace que la mayor parte de la información que he encontrado esté en inglés y muy dispersa. El conocimiento del idioma de por si no debería ser impedimento y no lo ha sido, pero ha retrasado mucho el estudio debido a la dificultad que ha supuesto reciclar mis conocimientos del idioma, a eso se le añade la dificultad de aprender los conceptos necesarios para desarrollar el asistente.

Esas dificultades que he encontrado, una vez superadas, hacen que me sienta muy satisfecho de haber aprendido nuevas herramientas y técnicas que desconocía. Tras adquirir esos conocimientos ha crecido la inquietud de profundizar en esas herramientas y eso creo que es una reflexión muy positiva.

Anteriormente, he comentado que el período de aprendizaje ha sido más amplio del que suponía y eso ha ayudado a que el desarrollo de este proyecto haya ocupado más tiempo del inicialmente previsto. También ha habido varias replanificaciones con el consiguiente aumento del tiempo de desarrollo del proyecto. Las replanificaciones han sido debidas a cambios en la idea original y posteriores retoques según surgían dificultades en el plan previsto.

He conocido el metamodelado para el diseño y creación de sistemas estructurados, he trabajado el diseño de software dirigido por modelos, he aprendido como crear editores gráficos y he aprendido la creación de aplicaciones sobre la base de Eclipse, además de haber ampliado mi destreza en la implementación de código Java. El trabajo realizado en el conjunto del desarrollo del proyecto también ha resultado positivo para ampliar y consolidar los conocimientos que poseía.

Considerando todo lo anterior creo, que el desarrollo de este proyecto ha resultado satisfactorio para mí ya que he podido conocer nuevas técnicas y herramientas que asientan y amplían los conocimientos adquiridos durante la etapa en la universidad y que espero sean válidas en el futuro recorrido laboral.

7.2 Trabajo Futuro

La base de este proyecto está en adaptar el metamodelo de UIML a los intereses que nos conviene y sobre esa base es posible realizar mejoras y desarrollar otras vías de trabajo.

Los componentes de UIML que se han trabajado en este proyecto son:

1. Structure: Indica la estructura de la interfaz de forma abstracta.
2. Style: Relaciona propiedades con los elementos. En nuestro caso relaciona los recursos con los elementos.
3. Content: Establece el valor de las referencias, los recursos en nuestro caso.

Sería interesante trabajar con el elemento Behavior que permite definir reglas para modelar el comportamiento de los elementos. Siguiendo como base el método seguido en este proyecto, puede crearse un editor donde sea posible crear y definir reglas que modelen el comportamiento de los elementos y acoplarlo a este proyecto enriqueciendo

así el documento UIML final, donde se muestra la composición de la interfaz abstracta de usuario y los recursos asociados a los componentes de esa interfaz.

Otra posible mejora puede ser añadir una vista al editor de asociación para poder asociar las prioridades de los recursos de interacción relacionados con cada AIO.

Siguiendo este camino de ir trabajando con los componentes de un metamodelo UIML y dado que en este proyecto se han tratado solamente unos pocos componentes, hay una vía abierta para enriquecer el documento UIML de una manera que refleje fielmente las intenciones del diseñador del servicio ubicuo.

CAPÍTULO 8 BIBLIOGRAFÍA

[1] Aragón H. (2002) *UIML look and Feel: Lineamientos de Diseño para interfaces en UIML*. http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/aragon_c_h/

[2] Coyette A, Faulkner S, Kolp M, Limbourg Q, Vanderdonckt J. (2004) *SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on UsiXML*. In Palanque Ph, Slavik P, Winckler M, (eds) Procs. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004), ACM Press, New York, 2004, 75-82

[3] Gronback R, (2009) *Eclipse Modelling Project: A Domain-Specific Language Toolkit*. Addison-Wesley (ed).

[4] Guerrero-García J, González-Calleros JM, Vanderdonckt J, Muñoz-Arteaga JA. (2009) *Theoretical Survey of User Interface Description Languages: Preliminary Results*. In Procs. of Joint 4th Latin American Conf. on HCI-7th Latin American Web Congress LAWeb/CLIHC'2009, IEEE Computer Society Press, Los Alamitos, 2009, 36-43.

[5] Helms J, Schaefer R, Luyten, K, Vermeulen J, Abrams M, Coyette A, Vanderdonckt J. (2009) *Human-Centered Engineering of Interactive Systems with the UserInterfaceMarkupLanguage.* http://jozilla.net/uploads/Research/HelmsLuytenSchaeferVermeulenAbramsCoyetteVanderdonckt_hcse2009.pdf.

[6] ITEA2 Information Technology for European Advancement. http://www.usixml.eu/UIDL_2011.

[7] LiquidApps application. http://liquidapps.harmonia.com/features/

[8] Limbourg Q, Vanderdonckt J, Michotte B, Bouillon L, López V. (2004) *USIXML: A language supporting multi-path development of user interfaces*. In: Bastide et al. (eds) EHCI/DS-VIS. LNCS. 3425. Springer, 200-220.

[9] McAffer J, Lemieux JM, Aniszczyk C. (2010) *Eclipse Rich Client Platform second edition*. Addison-Wesley (ed).

[10] Meskens J, Vermeulen J, Luyten K, Coninx K. (2008) *Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me*. In Procs. of the working conference on Advanced Visual Interfaces, AVI 2008 (Napoli, Italy, May 28-30, 2008), 233-240.

[11] Montero F, Lozano MD, González P. (2005) *IdealXML: an Experience-Based Environment for User Interface Design and pattern manipulation*, Technical report DIAB-05-01-4, University of Castilla-La Mancha, Albacete, 24 January 2005.

[12] Montero F, López-Jaquero V. (2010) *Guilayout++: Supporting Prototype Creation and Quality Evaluation for Abstract User Interface Generation*. In Procs. of the 1st. USer Interface eXtensible Markup Language Workshop (UsiXML-EICS 2010). June 20, 2010, Berlin, Germany. pp 39-44.

[13] Mori G, Paterno F, Santoro C. (2002) *CTTE: Support for developing and analyzing task models for interactive system design*. IEEE Trans. on Soft. Eng. 28, 797–813.

[14] Riba N. (2007) *Un enfoque MDA para el desarrollo de aplicaciones basadas en un modelo de componentes orientados a servicios* (pp 17, 25). Universidad Autónoma Metropolitana,Unidad Iztapalapa.

[15] Steinberg D, Budinsky F, Paternostro M, Merks E. (2008) *EMF: Eclipse Modelling Framework, second edition*. Addison-Wesley (ed).

[16] Vogella L. (2012) *Eclipse RCP Tutorial*.
http://www.vogella.com/articles/EclipseRCP/article.html

[17] Xalan http://xml.apache.org/xalan-j/

[18] Xerces Java Parser, http://xerces.apache.org/

[19] XSLT recommendation, XSL Transformations, http://www.w3.org/TR/xslt/

[20] XML recommendation eXtensible Markup Language, http://www.w3.org/XML/

CAPÍTULO 9 ANEXOS

A.-Glosario

Interfaz abstracta: Representación de la interfaz de usuario común a cualquier dispositivo y cualquier plataforma tecnológica.
Middleware: Software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos.
Servicio ubicuo: Tener un servicio disponible desde cualquier lugar, en cualquier momento y de forma ininterrumpida.
Target: Dispositivo a controlar. Cada uno aporta uno o varios ”UIS” (User Interface socket) a través de los cuales el URC puede acceder al estado del Target y a controlar el Target.
Tecnología UCH: Universal Control Hub. Arquitectura de software estandarizada de accesibilidad e interoperabilidad que persigue que todos los sistemas y dispositivos presentes en el hogar digital puedan ser controlados desde el dispositivo personal más cercano a las necesidades de cada usuario.
URC: Universal Remote Control. Estandar internacional ISO/IEC que permite crear un marco global de accesibilidad universal a través de la definición de una arquitectura entre una consola remota y un target (objetivo), y la definición de la comunicación entre ellos basada en una serie de documentos XML.

Asistente para la generación de interfaces de usuario abstractas en entornos ubicuos:
Anexos

B.-Manual de usuario

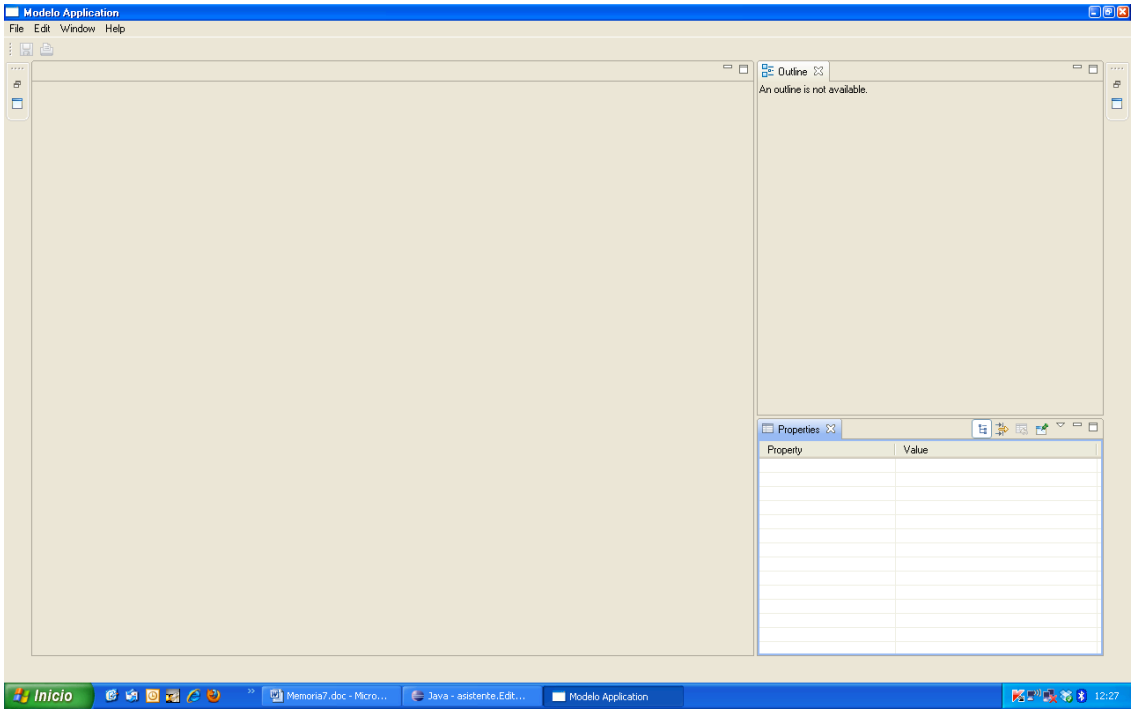

Figura 37: Interfaz Principal

Esta interfaz presenta el editor que todavía se encuentra vacío, al seleccionar el documento XML del servicio ubicuo se mostrará la paleta de herramientas y la disposición de los objetos abstractos de interacción que se ha identificado en el documento seleccionado.

La interfaz también presenta las dos vistas que se aprecian en la parte derecha de la figura 37. La vista "Outline" es una vista de resumen y es una forma rápida de ver qué métodos y atributos se encuentran definidos dentro de una clase Java. La vista "Properties" que se encuentra debajo de la anterior muestra las propiedades del elemento seleccionado en el editor, esa información puede ser editada por el diseñador para proporcionar los valores a los atributos del elemento seleccionado.

La barra de menú del workbench nos ofrece cuatro opciones. Solamente nos fijamos en la pestaña "File" (ver figura 38). Abriendo esta pestaña obtenemos distintas opciones para seleccionar, escogemos la opción "Selecciona un servicio…" que permite seleccionar el archivo de servicio que nos interesa.

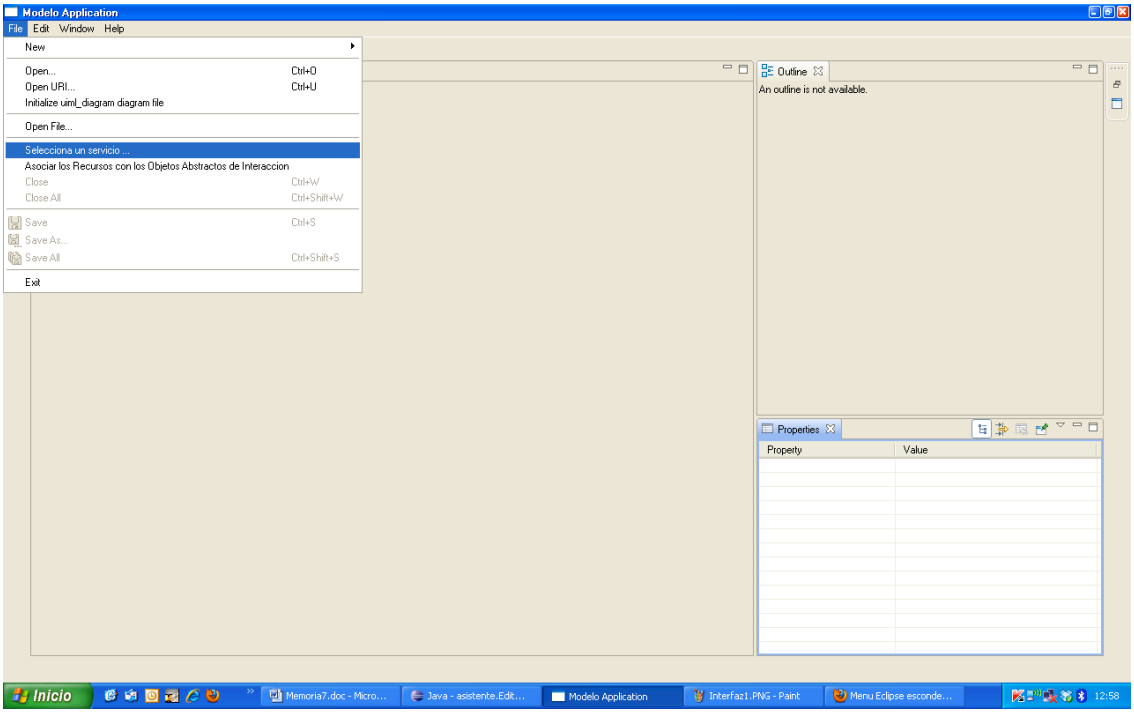

Figura 38: Selección del servicio

Como se puede observar en la figura 39 tras escoger esta opción, un asistente nos ayuda en el proceso de elegir la ruta donde se encuentra el fichero que nos interesa.

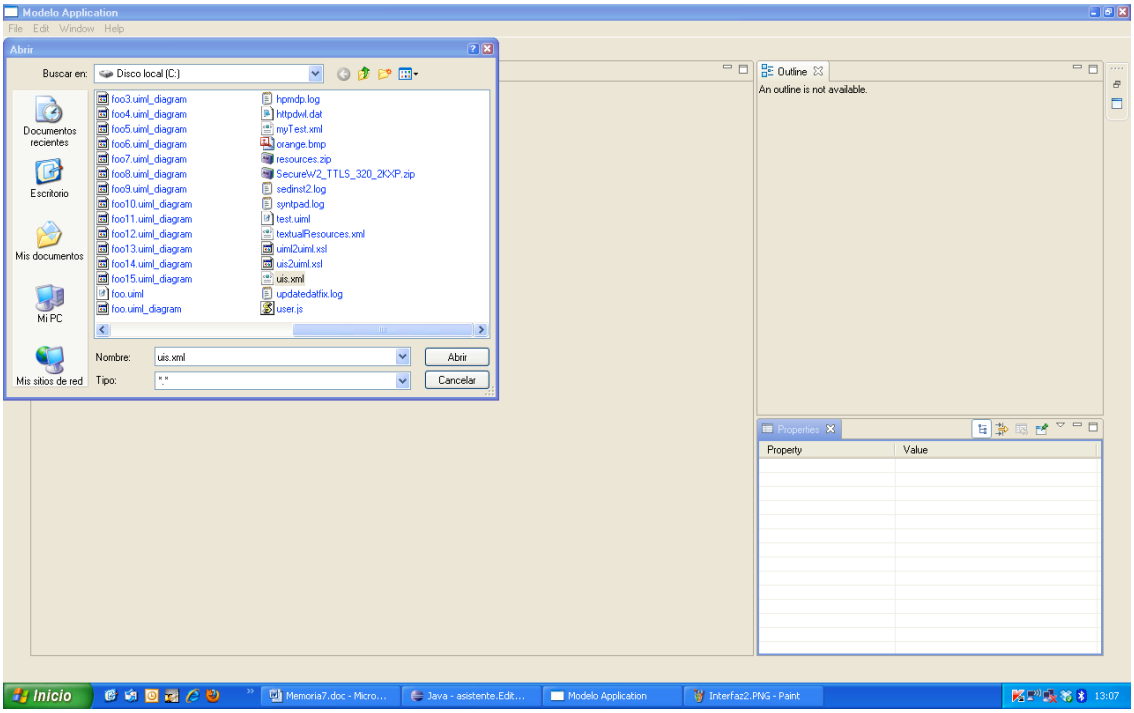

Figura 39: Selección del fichero de servicio

Seleccionamos la ruta del fichero y pulsamos “Abrir”. Tras seleccionar el fichero del servicio nos aparece otra interfaz donde seleccionamos el fichero (.uiml_diagram) que guardará la configuración de la interfaz abstracta que se creará, seleccionamos la ruta y el nombre del fichero y pulsamos “Next”, esta acción se contempla en la figura 40.

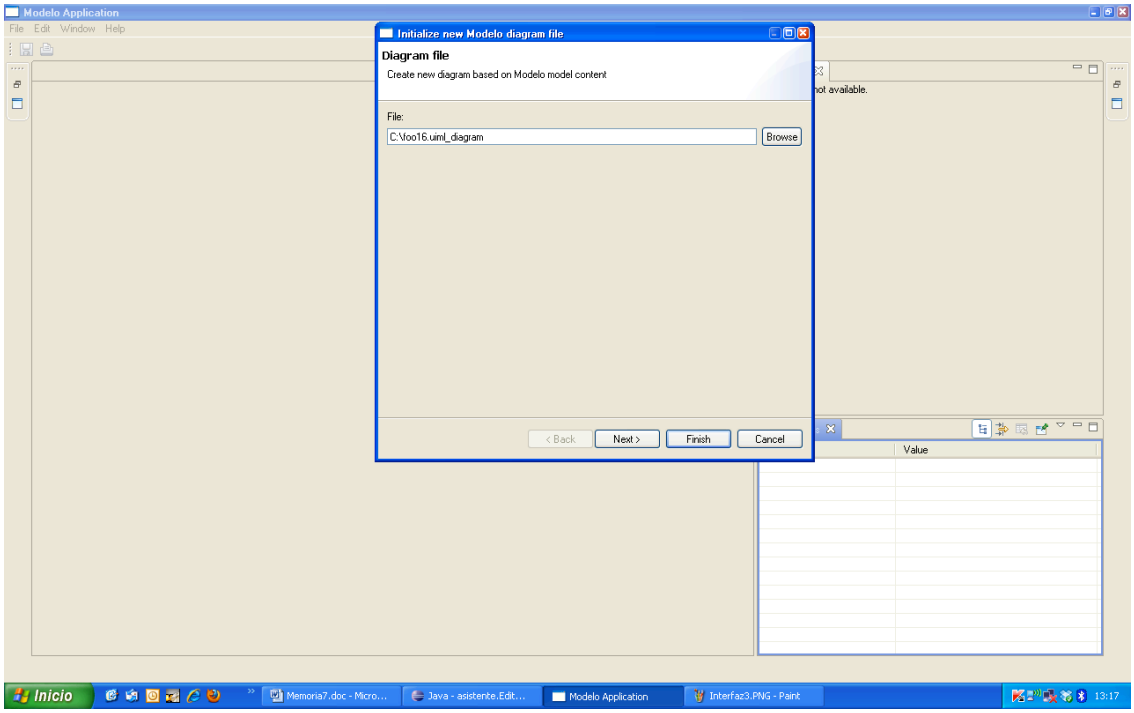

Figura 40: Selección de la ruta del fichero descriptivo de la interfaz

Tras seleccionar la ruta del fichero de salida, en la siguiente pantalla pulsamos “Finish” y se muestra en el editor la configuración de la interfaz abstracta que define el archivo de servicio seleccionado.

Esta interfaz abstracta presentada por el sistema que se aprecia en la figura 41, puede ser reformada en los aspectos que el diseñador estime oportuno, puede eliminar elementos mostrados en el editor, incluir nuevos elementos, redimensionar las figuras, cambiar el identificador de las figuras o aceptar la interfaz presentada si lo cree conveniente.

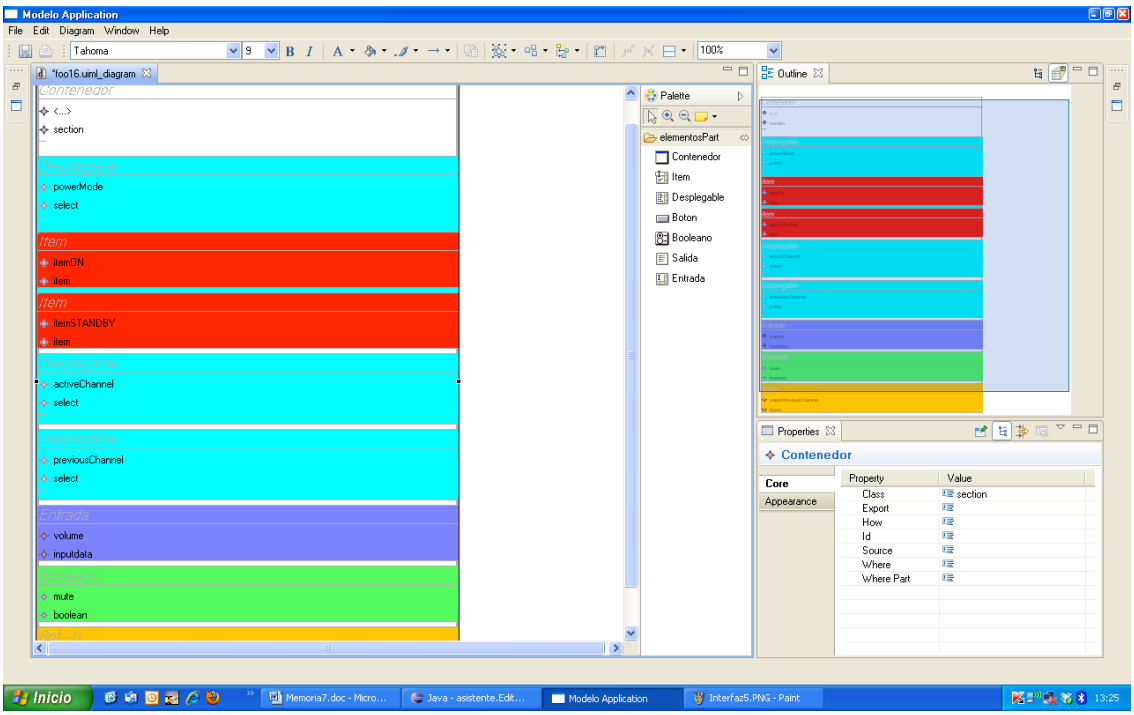

Figura 41: Interfaz presentada por el sistema

Para borrar una figura del lienzo del editor seleccionamos la misma con el ratón y pulsando el botón derecho del mismo elegimos la opción “Delete from Model”.

Si queremos mover la figura sobre el lienzo del editor la seleccionamos con el ratón y arrastramos la figura al lugar conveniente.

La paleta del editor nos ayuda a seleccionar el tipo de objeto abstracto de interacción que queremos incorporar al lienzo. Pasando el ratón sobre los iconos de la paleta se muestra información sobre el tipo de objeto abstracto de interacción que representa cada uno.

Los elementos de la paleta se seleccionan con el ratón y se arrastran a la posición que interesa. No todos los elementos de la paleta pueden colocarse directamente sobre el lienzo del editor ni dentro de otros elementos, esa información se recibe mientras se arrastra el elemento. Mientras es posible colocar el elemento se nos muestra el símbolo de arrastre y si no lo es observamos el símbolo de prohibido, facilitándonos la labor de composición de la interfaz abstracta.

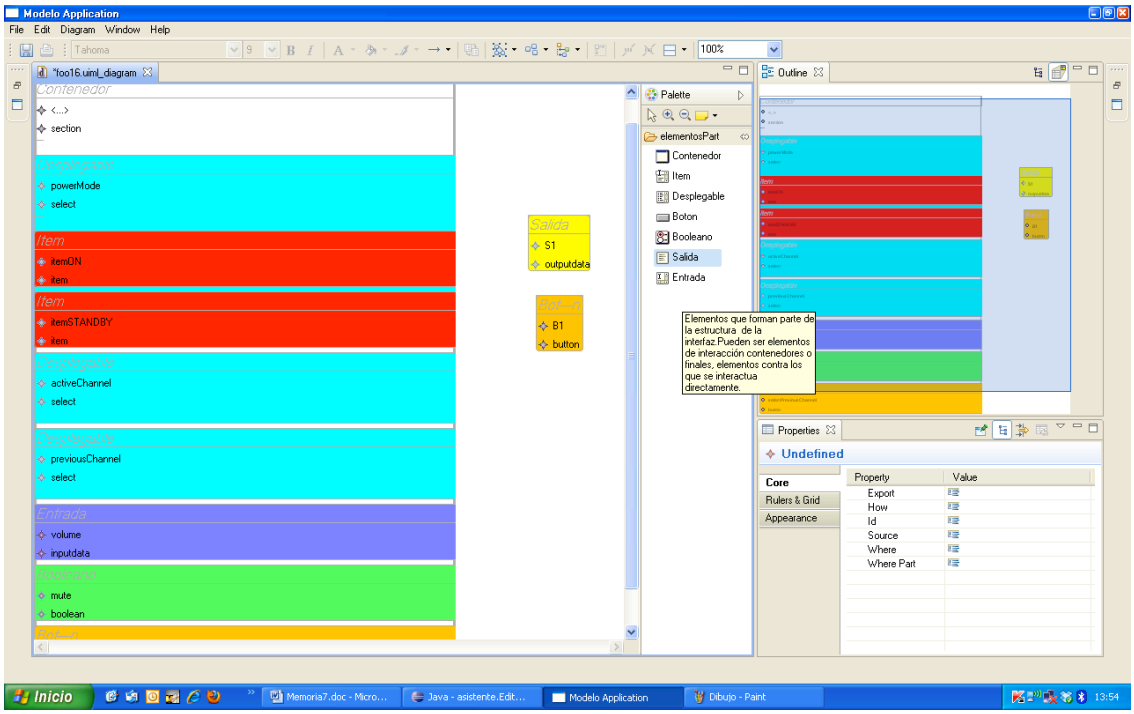

Figura 42: Editor en uso

En la figura 42 se observa el texto informativo y los elementos incorporados a la composición presentada por el sistema.

Tras realizar la composición, para realizar la asociación de recursos a los elementos de la interfaz abstracta, seleccionamos la pestaña “File” y entre las opciones que muestra escogemos “Asociar los recursos con los Objetos Abstractos de Interacción”.

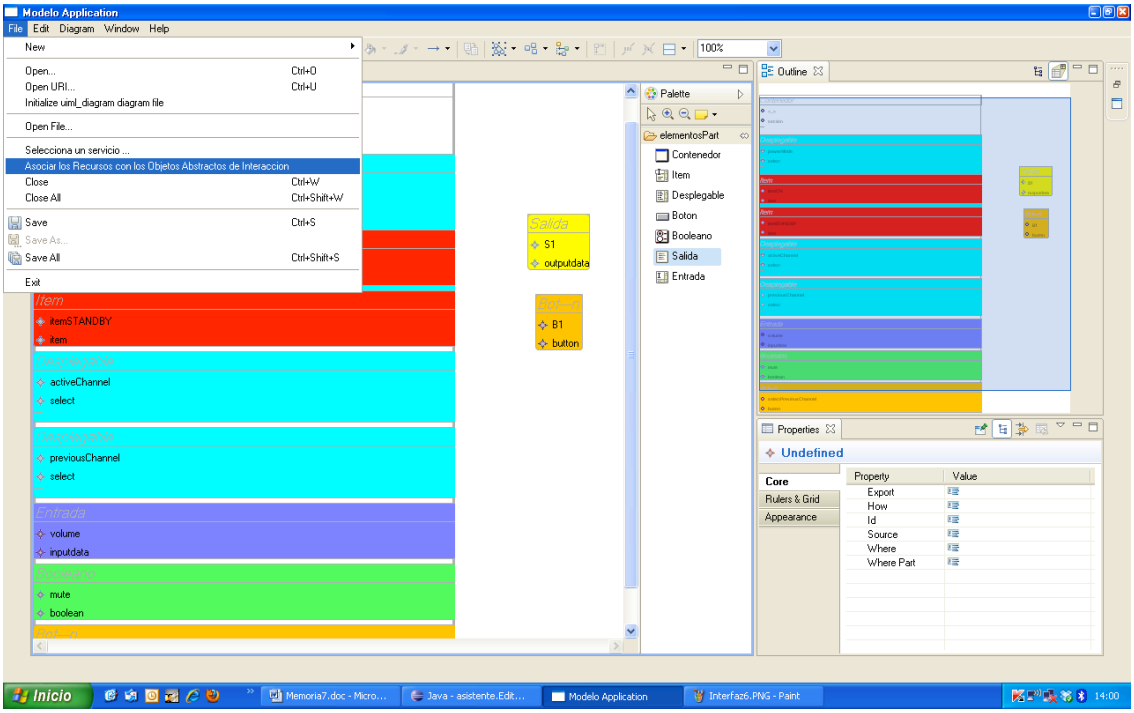

Figura 43: Asociar los recursos

Seleccionando esta opción, que se observa en la figura 43, si hemos cambiado la composición inicial el sistema nos pide confirmar los cambios, se muestra la vista del módulo de asociación que ofrece los objetos abstractos de interacción que componen la interfaz abstracta anteriormente diseñada.

En la figura 44 se pueden observar los objetos abstractos de interacción anteriormente seleccionados y los dos nuevos elementos que se han añadido a la composición inicial.

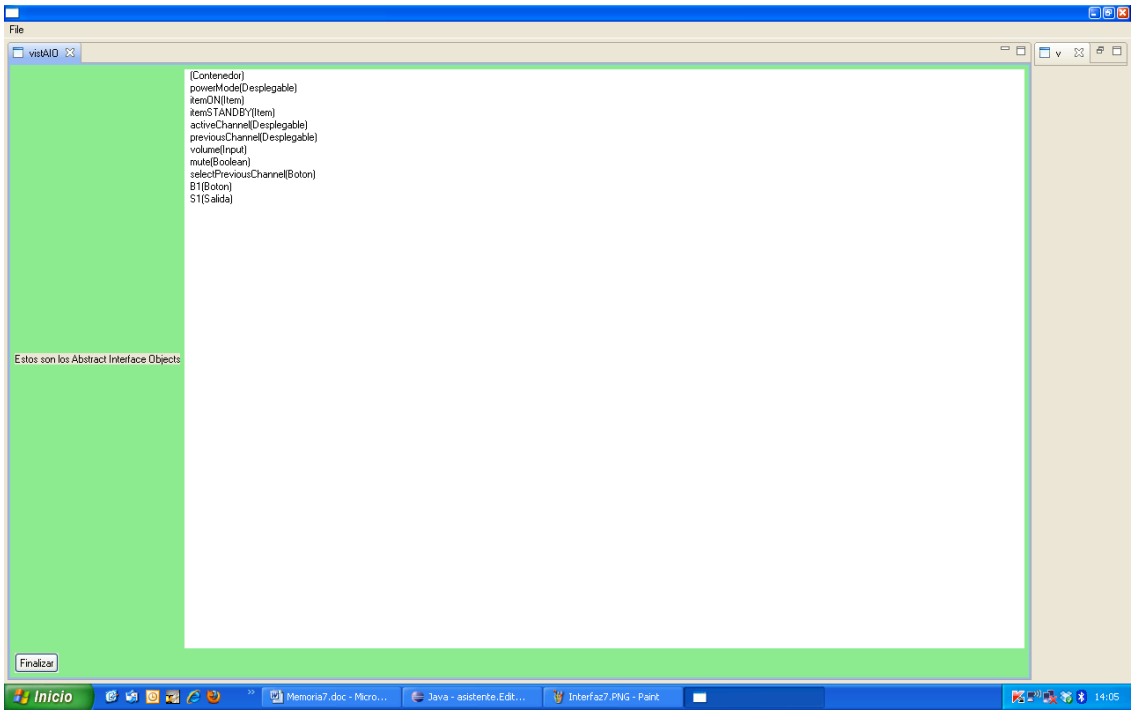

Figura 44: Vista AIO

Seleccionando uno de los AIO que se muestran aparece otra vista con los recursos disponibles para asociarlos a los AIO, esta vista se ve en la figura 45.

En esta vista de recursos se muestran los mismos en un árbol, que muestra los recursos agrupados según el tipo de recurso.

Podemos seleccionar los recursos que estimemos oportuno y cuando queramos guardar las asociaciones pulsamos “Guardar/Volver”, se guardan las asociaciones y volvemos a la vista de los AIO donde podemos escoger otro AIO y realizar las asociaciones oportunas.

Si no estamos interesados en las asociaciones que estamos creando pulsamos “Cancelar” y volvemos a la vista de los AIO, en este caso las asociaciones creadas son rechazadas.

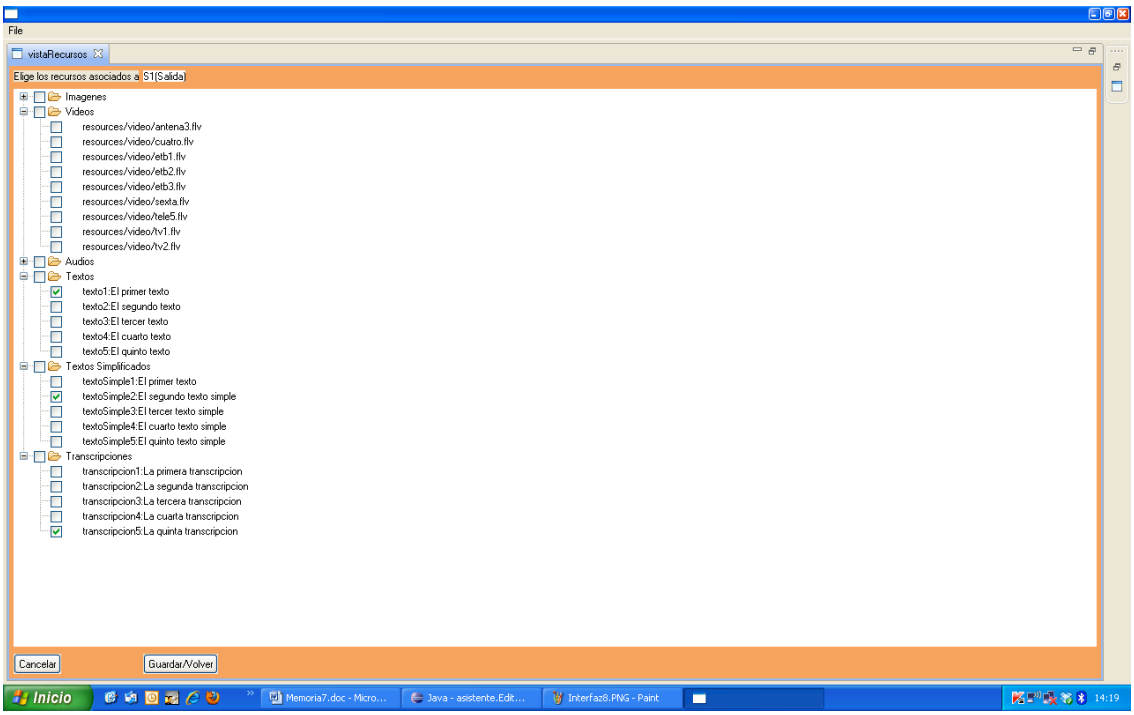

Figura 45: Vista de recursos

Las casillas correspondientes al tipo de recurso que incorporan el icono de carpeta no son seleccionables, aunque sean marcadas no son recogidas esas selecciones.

Tras realizar las asociaciones que se necesitan, la utilización de la herramienta gráfica puede terminarse de dos formas diferentes, pulsando “Finalizar” o escogiendo la opción “Exit” de la pestaña “File”.

Tras usar la herramienta gráfica, el sistema genera un documento UIML acorde con Egoki donde se refleja la composición de la interfaz abstracta y los recursos asociados a sus componentes.

Asistente para la generación de interfaces de usuario abstractas en entornos ubicuos: Anexos