MovilTurismo: Plataforma de creación de aplicaciones para móviles basada en la generación de código por modelos

Beatriz Marchante Banegas

Trabajo Fin de Máster (TFM), Curso 2010 - 2011
Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información
Universidad Politécnica de Valencia
Cno. de Vera s/n, 46071 Valencia, España
beamarba@eui.upv.es

Director: Joan J. Fons i Cors
CoDirector: Ismael Torres Boigues

Septiembre 2011

Resumen

El incremento del uso de los teléfonos inteligentes o smartphones en la vida cotidiana es un hecho que podemos comprobar con sólo mirar a nuestro alrededor. Día a día, se intentan mejorar estos dispositivos para garantizar una mayor funcionalidad y portabilidad que nos permita gestionar nuestras actividades diarias.

Hoy por hoy la creación de dicha funcionalidad recae sobre los programadores software, que son los encargados de crear las aplicaciones. Sin embargo empieza a destacar el desarrollo conocido como *End-User* development cuyo objetivo es permitir a los usuarios finales crear aplicaciones sin necesidad de poseer conocimientos sobre programación. Ellos son los que realmente conocen sus propias necesidades de gestión.

Para llevar a cabo este propósito, usaremos la arquitectura conocida como MDA. Esta opción de desarrollo fue concebida para dar soporte a la ingeniería dirigida por modelos de los sistemas software. El desarrollo de Software Dirigido por Modelos constituye una aproximación para el desarrollo de sistemas software basada en la separación entre los conceptos o abstracciones de los sistemas (representados mediante modelos), y las soluciones o dependencias tecnológicas de los sistemas.

En pos de aprovechar esta tecnología, en este trabajo se ha desarrollado *MovilTurismo*, una plataforma de desarrollo de aplicaciones móviles basada en la generación de código por modelos. Inicialmente concebida para desarrollar guías de soporte a actividades turísticas, pero que a lo largo del trabajo realizado ha evolucionado a una plataforma más genérica.

Esta plataforma permitirá a los usuarios registrados en ella realizar su propia aplicación sin tener ninguna experiencia en programación y les permitirá aprovechar la tecnología existente de manera que podrán cubrir sus necesidades de negocio por ellos mismos.

Abstract

The increased use of Smartphone's in everyday life is a fact that we can see just by looking around us. Day by day, try to improve these devices to ensure greater functionality and portability that allows us to manage our daily activities.

Today, the creation of such functionality lies with the software developers who are responsible for creating applications. But the development comes to the fore known as *End-User Development* whose aim is to enable end users to create applications without having programming knowledge. They are the ones who really know their own management needs.

To accomplish this purpose, we use the architecture known as MDA. This development option was designed to support model-driven engineering of software systems. The development of Model-Driven Software is an approach to developing software systems based on the separation between the concepts or abstractions of systems (represented by models), and technological solutions and systems departments.

In pursuit of leverage this technology, this paper has developed *MovilTurismo*, a platform for developing mobile applications based on code generation by model. Initially conceived to develop guidelines to support tourism, but throughout the work has evolved into a generic platform.

This platform will allow registered users in it implement its own application without any programming experience and allow them to leverage existing technology so they can meet their business needs by themselves.

Tabla de contenido

Lista de figuras

Lista de Tablas

1. Introducción

El incremento de la demanda de aplicaciones móviles, la necesidad de disponer aplicaciones móviles para diferentes dominios y la propia evolución de la tecnología que les da soporte, requiere automatizar el proceso de producción del software, además de hacer que dicho proceso sea flexible, ágil y que permita tratar la parte variable de la aplicación de la manera más robusta posible.

En la actualidad, está muy difundido un enfoque basado en la reutilización de código existente usado por otros desarrolladores de aplicaciones similares, mediante Frameworks y APIs ad-hoc. Este modelo está muy limitado y no es realmente escalable o fácilmente adaptable a nuevos entornos o requisitos.

Entre sus desventajas, podemos destacar que es propenso a generar errores, y aunque uno de sus beneficios es la mejora de la productividad, uno de los problemas principales es que fomenta que los desarrolladores se centren en aspectos tecnológicos de la solución en vez de a aspectos creativos y de diseño abstracto de sistemas. Además, esta manera de trabajar impide transferir experiencias entre desarrolladores y grupos de éstos.

Así pues, para llevar a cabo un proceso de desarrollo automatizado deben existir herramientas, lenguajes de modelado y arquitecturas que nos permitan partir de la conceptualización de las necesidades de negocio y que nos lleven a la implantación bajo tecnologías específicas

En este marco, destacamos por ejemplo UML como lenguaje de modelado usado en algunas de estas herramientas y que les ayuda a generar código a partir de un Modelo Conceptual.

Como arquitectura principal, es de obligado cumplimiento mencionar MDA (Model-Driven Architecture) como un acercamiento al diseño de software propuesto por el Object Management Group 1(OMG). Fue concebido para dar soporte a la ingeniería dirigida por modelos de los sistemas software y proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos.

MDA propone el uso de lenguajes abstractos para usar como modelos en la especificación de sistemas. UML es un lenguaje de uso general y destaca la independencia de la plataforma, si bien, eso no puede ser absoluto puesto que las características propias de la tecnología, influirán en las restricciones del modelo y que se usa de manera masiva en el desarrollo de software dirigido por modelos.

MDA asume que 3 tipos de modelo son suficientes:

- Computation-independent model
- Platform-independent model
- Platform-specific model.

Se enfoca en transformaciones y es difícil lograr un proceso completamente automático que vaya de concepto a implementación.

La metodología debe incluir el manejo de la parte variable que no se puede automatizar y los cambios que se requieran durante el mantenimiento de una aplicación.

El desarrollo de Software Dirigido por Modelos constituye una aproximación para el desarrollo de sistemas software basada en la separación entre la especificación de la estructura y funcionalidad esenciales del sistema y la implementación final, usando plataformas de implementación específicas. [2]

Se persigue elevar el nivel de abstracción en el desarrollo de software dándole una mayor importancia al modelado conceptual y al papel de los modelos en el desarrollo de software actual.

El uso de notaciones de modelado conceptual orientadas a objeto para la especificación de la estructura y funcionalidad del sistema, permite especificar la manera en la que el usuario final interactúa con el sistema para proveer y obtener datos e información.

La especificación completa y precisa de los aspectos estáticos, dinámicos y de presentación del sistema de información en los modelos conceptuales permite la posterior generación automática del código de los sistemas. El proceso de generación automatizado constituye un paso importante para la industria de producción del software.

[1] www.omg.org/

1.1. Motivación

Hoy en día, el uso de los Smartphones está totalmente integrado en la vida cotidiana.

Mientras se procura reducir su tamaño en pro de una mayor portabilidad, su funcionalidad aumenta, convirtiéndolos en dispositivos inteligentes con capacidad de cómputo añadido, siendo sólo limitada por las aplicaciones que en ellos se instalen. [3]

El uso de estos dispositivos móviles permite gestionar la información que consideramos más esencial para realizar nuestras actividades diarias, con la ventaja de tenerla siempre a mano en cualquier momento.

Paralelamente a todas las ventajas que se pueden encontrar en los dispositivos móviles también se encuentran desventajas, entre ellas las siguientes [3]:

- Pantallas pequeñas.
- Batería limitada y menor velocidad de procesamiento.
- Ancho de banda variable

Aunque son los programadores los poseedores de los conocimientos necesarios para crear las aplicaciones, son los usuarios finales quienes conocen realmente sus propias necesidades de gestión.

Los usuarios finales [4], no solo son usuarios “informales”, “novatos” o “inocentes”, pueden ser químicos, bibliotecarios, arquitectos, profesores o contables que tienen necesidades computacionales distintas y necesitan hacer uso de ellas. No obstante, estos usuarios no están interesados en convertirse en programadores profesionales.

Es aquí donde entra en juego el desarrollo conocido como *End-User Development*. Este desarrollo se define [4] como un conjunto de actividades o técnicas que permiten a personas que no son desarrolladores de software profesionales, crear o modificar un artefacto software.

En *End-User Development* muchas técnicas tienen como objetivo permitir a los usuarios finales programar o personalizar su sistema y por tanto, se persigue una alineación natural entre las expectativas del usuario final y las capacidades del sistema.

En resumen, la motivación reside en centrarse en los usuarios finales, tanto por ser conocedores mejor que nadie de sus propias necesidades, como por potenciar su capacidad de crear y personalizar sistemas.

1.2. El problema

Como hemos mencionado anteriormente, es el usuario final quien conoce al 100% sus necesidades y que necesita en un dispositivo móvil. Sin embargo es el programador el que tiene el conocimiento necesario para llevar a los dipositivos móviles estas necesidades.

Al crear soluciones genéricas, es más que probable que algunas de las necesidades no sean cubiertas y el usuario deba ponerse a programar para generar el sistema o solicitarlo "a la carta" a alguna empresa que se dedique a ello.

Además hay que añadirle el gasto que supone generar la aplicación y el tiempo necesario para llevarla a cabo, así como satisfacer todas las necesidades del cliente.

¿Puede este escenario cambiar? En el siguiente apartado, mostramos la solución propuesta y cómo se intenta mejorar esta situación a través de la aplicación MovilTurismo.

1.3. Solución Propuesta

MovilTurismo es una plataforma Web concebida para permitir a los usuarios finales crear aplicaciones móviles a su gusto sin necesidad de conocimientos previos de programación.

Una plataforma [27] Web permite relegar la ejecución de los programas a la propia red, evitándonos así los problemas que supone la portabilidad a diferentes sistemas en los que puede ejecutarse una aplicación. Usar la Web para nuestro propósito, nos hace también obviar las instalaciones, los requisitos mínimos, las actualizaciones, etc. en el *cliente*.

Únicamente es necesaria una conexión a Internet y un dispositivo de acceso (ordenador, móvil, etc.).

Dadas las ventajas que supone la tecnología MDA y puesto que aparentemente ninguna de las soluciones encontradas en la red (ver capítulo 2: Estado del Arte) usa esta tecnología, MovilTurismo intenta solventar el escenario propuesto usándola.

Permitirá a los usuarios generar las aplicaciones para cualquier plataforma disponible y les dejará la opción de personalizar la aplicación. Para ello solo será necesario un navegador compatible con HTML5.

En definitiva, la solución propuesta en la presente tesina es una combinación entre el desarrollo End-User y la arquitectura MDA.

Para conocer en profundidad la aplicación, leer los capítulos 4 y 5 del presente documento.

1.4. Contexto

Esta tesina de Máster ha sido desarrollada bajo el contexto del Centro de Investigación en Métodos de Producción de Software (PROS) de la Universidad Politécnica de Valencia ubicado en el departamento DSIC.

Vemos a continuación las asignaturas impartidas en el Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información que se aplicaron de manera directa o indirecta en el desarrollo del presente trabajo de tesis:

- *Desarrollo de aplicaciones en Java*: El contenido de esta asignatura fue de especial relevancia durante la implementación, ya que la capa de negocio se programa en Java.

- *Integración Semántica de Datos:* Fue útil para la definición del modelo de datos.

- *Calidad de Sistemas de Información:* Para el modelado de la interfaz de usuario de las aplicaciones web y móviles.

- *Diseño de Aplicaciones basadas en dispositivos móviles:* Su contenido trata de crear una aplicación totalmente funcional para dispositivos con Windows Mobile.

- *Técnicas Avanzadas en Ingeniería de Requisitos:* Útil para la creación de los casos de uso, así como la funcionalidad de la aplicación.

- *Patrones Software y Generación de Código:* Esta asignatura, fue el pilar sólido de la presente tesina. En esta asignatura es donde se creó el prototipo que sirvió como base a MovilTurismo a través del framework MosKitt.

1.5. Estructura de la tesis

El documento está organizado en 10 capítulos. El capítulo 2 presenta un recopilatorio de las diferentes soluciones que nos proporciona la red y que aportan su granito de arena a la generación automática de código.

El capítulo 3 presenta el contexto tecnológico empleado en la presente tesina, así como las bases para su creación y la tecnología empleada. Por otro lado, el capítulo 4 muestra qué es la plataforma y el capítulo 5, la creación de la misma los pasos seguidos, funcionamiento y consideraciones necesarias para el desarrollo, así como casos de uso y algunos diagramas.

Siguiendo con el capítulo 6, veremos el caso de estudio que se ha abordado y el capítulo 7 muestra el manual de usuario de la aplicación,. Siguiendo con el punto 8, las conclusiones del trabajo realizado y en el 9 algunas directrices a seguir en un trabajo futuro.

A continuación, el capítulo 10 presenta el listado con las referencias a los trabajos y las webs consultadas durante la realización de esta tesis.

Por último, encontramos 4 anexos en los que se encuentra la siguiente información: modelo de casos de uso, metamodelo, script de creación de base de datos y contenido de las plantillas de generación.

2. Estado del arte

En la actualidad, sería imposible pensar en los móviles actuales sin las grandes aplicaciones que éstos incluyen. Existe gran variedad de aplicaciones móviles que buscan hacerse hueco en el mercado: Se trata de una evolución continua y las grandes compañías pujan por obtener el mayor beneficio posible. Y no es para menos ya que se cuenta con aproximadamente 400000 aplicaciones en la App Store [5] y 250000 en el Android Market [6].

En este marco, no es de extrañar que exista una gran cantidad de empresas que se dediquen en exclusiva a la creación de aplicaciones móviles a la carta para cualquier usuario / empresa que las solicite.

En este mercado en expansión es necesario mencionar la creciente aparición de aplicaciones web y de empresas, dedicadas a proporcionar una infraestructura tecnológica para que sea el propio usuario quien llene de contenido y configure la estructura de su propia aplicación para móvil.

El objetivo de esto es que el usuario no tenga que ponerse a programar, ni aprender a hacerlo, si no, que siga una interfaz intuitiva para crear su propia aplicación sin realizar grandes esfuerzos. En este punto, podemos mencionar el desarrollo End-User como un acercamiento entre los artefactos de desarrollo y los usuarios finales, con el fin de hacerlo mas intuitivo para aquellos usuarios que no tengan los conocimientos necesarios.

¿Cómo es esto posible? Una de las tecnologías que hace posible que esto se pueda llevar a cabo es el uso del Model Driven Development (MDA).

Tal y como se ha mencionado anteriormente, usando MDA es posible minimizar el coste de desarrollo de nuevas aplicaciones así como mejorar la calidad del propio código, evitar errores y optimizar un proceso de desarrollo que en ocasiones resulta fatigoso.

Tras una búsqueda exhaustiva por la red, hemos encontrado soluciones similares a las que se proponen en el presente trabajo, las cuales mencionaremos a continuación. Pero antes, veamos la estructura que sigue el contenido que vamos a abordar:

El presente capítulo se organiza de la siguiente manera: en el punto 2.1 hacemos referencia a la aplicación **MovilEvento**, siendo esta aplicación un gran referente para la creación de MovilTurismo. En el punto 2.2 conoceremos la apuesta propia de Google: **App Inventor**, para la creación de las aplicaciones móviles y el punto 2.3, la aplicación **Buzztouch**. La generación de código de esta aplicación se basa en crear el código fuente que el usuario final podrá compilar y subir a la tienda de aplicaciones correspondiente. Como veremos más adelante una parte de MovilTurismo sigue estas bases para realizar la generación. El punto 2.4, muestra la empresa **Tekora** y su solución para poder ejecutar su aplicación en cualquier móvil y el punto 2.5 muestra **PhoneGap** un Framework multiplataforma. Finalmente el apartado 2.6 incluirá las conclusiones y algunas comparativas.

2.1. MovilEvento

MovilEvento[7] es una aplicación web que permite al usuario final, crear una aplicación basada en eventos (conferencias, convenciones, etc.…) para que los asistentes puedan instalársela en sus móviles y tengan siempre accesible la información. Está orientado a la creación de aplicaciones para móviles Nokia.

Las aplicaciones generadas pueden contener las siguientes funcionalidades:

- •Agenda
- •Direcciones
- •Mapas
- •Recomendaciones (ocio, turismo, etc.)
- •Horarios
- •Teléfonos de interés
- •Información comercial
- •Localizador de franquicias

Una vez el usuario se valida en el sistema, puede crear una nuevo “MovilEvento”, gestionar los datos de su cuenta de usuario o editar las guías que tenga creadas, etc.…

Sin embargo nos centraremos en la parte que concierne a la generación de la propia aplicación. Una vez que el usuario registrado ha creado el nuevo evento, accede a la siguiente interfaz, conocida como la zona de edición:

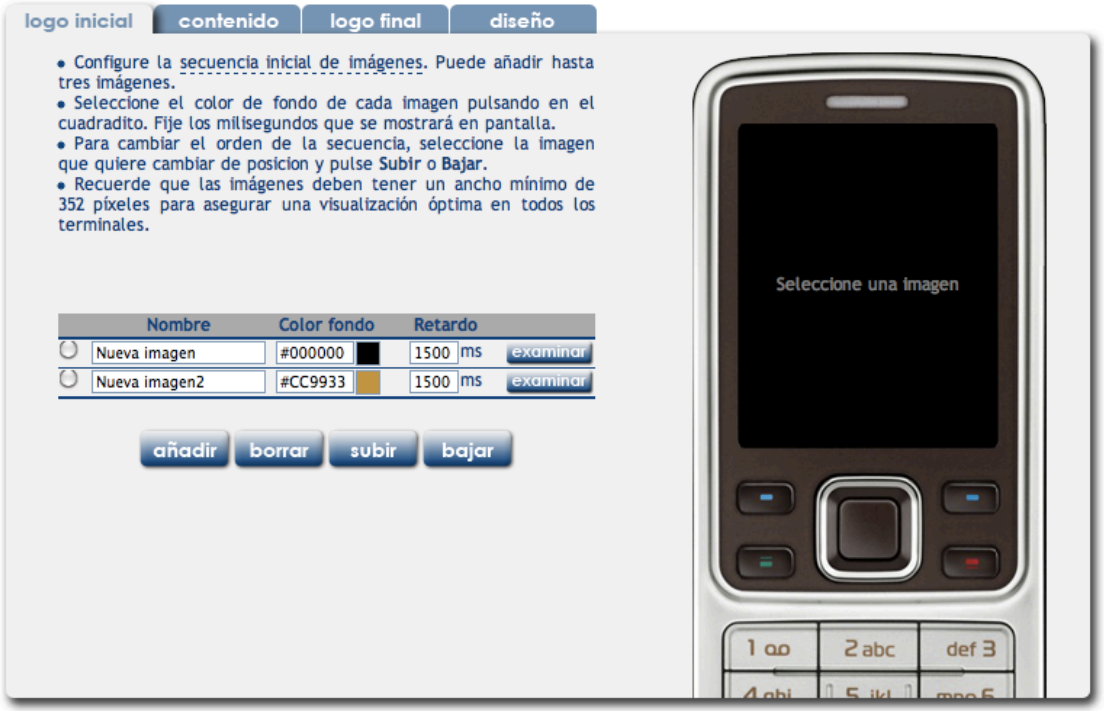


Figura 1: Zona de edición

En la parte superior, aparecen las opciones más genéricas y en la zona central, una serie de pestañas que permitirán al usuario ir rellenado los datos que contendrá la aplicación en función de las diferentes zonas:

- Logo Inicial
- Contenido
- Logo final
- Diseño

En la Figura 1 podemos ver la zona *Logo Inicial* donde, tras una breve explicación del funcionamiento, se solicita al usuario que introduzca la / las imágenes que aparecerán en el logo de la aplicación y el tiempo que permanecerán en pantalla. El funcionamiento del apartado Logo Final, es el mismo

Una vez tenemos la pantalla principal de la aplicación configurada, podemos pasar a crear su contenido. Para ello, el usuario debe ubicarse en la pestaña *Contenido*

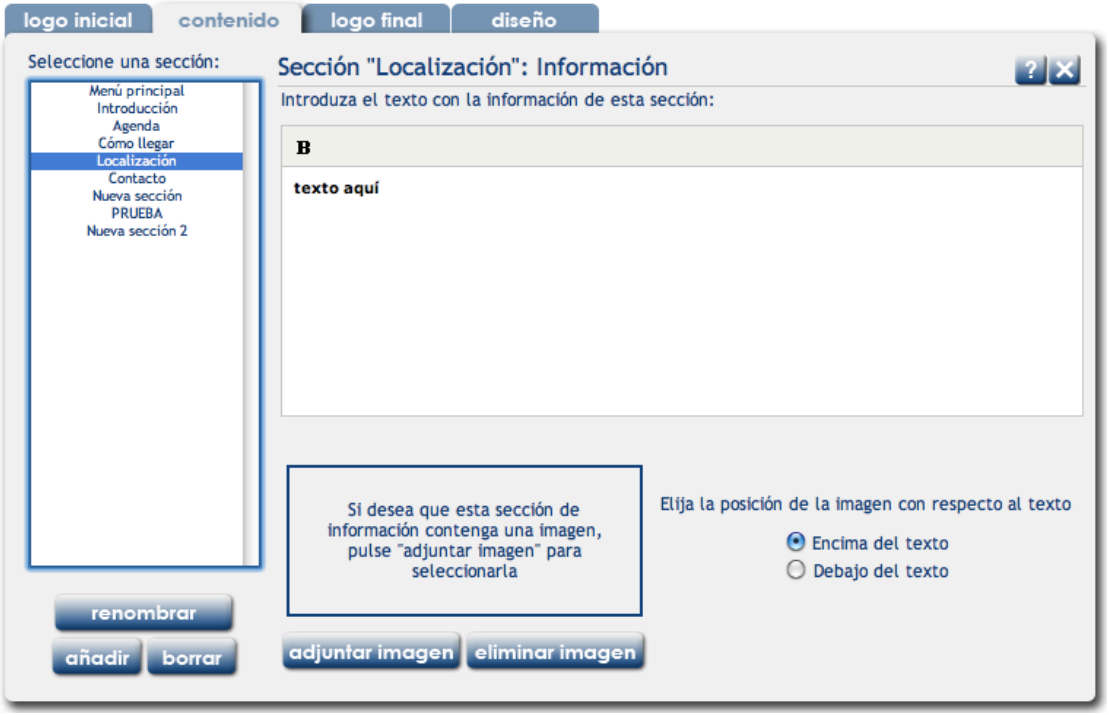


Figura 2: Edición del Contenido

El funcionamiento es el siguiente: el usuario seleccionará de la parte izquierda de la pantalla la sección que quiere completar. La aplicación trae una serie de secciones creadas por defecto, pero el usuario puede crear las suyas propias pulsando sobre el botón "Añadir", localizado debajo de la lista.

Cuando se pretende añadir una sección nueva, la aplicación nos solicita el nombre de la sección y nos invita a elegir de qué tipo será. Existen 4 tipos de secciones (no ampliables) con las que debe jugar el usuario: Información, Agenda, Mapa y Menú de Opciones.

En base al tipo seleccionado, irán apareciendo en pantalla diferentes métodos para completar la información. Como ejemplo ilustrativo: si el usuario ha creado una pantalla de tipo "Mapa", la aplicación le solicitará que adjunte la imagen que se usará como mapa. Si por el contrario, el usuario ha creado una sección de tipo "Información", aparecerá un cuadro de texto apto para que el usuario escriba en él lo que desee. Este último ejemplo se encuentra visible en la Figura 2.

De esta manera, el usuario va completando la aplicación poco a poco, sin necesidad de programar ni una sola línea de código.

Adicionalmente se le da la oportunidad de dar cierto formato a las pantallas que ha creado, mediante la elección, sobretodo, de los colores que tendrán los distintos tipos de letras, así como el color de fondo de la aplicación.

Para llevar a cabo la generación, usan una tecnología [8] propia que recibe el nombre de "Caponate". Dicha tecnología consiste, a groso modo, en definir un XML con el contenido de la aplicación y generarla a partir de éste.

2.2. App Inventor Beta

App Inventor[9] es una Beta de Google que permite al usuario final, desarrollar aplicaciones para móviles Android. A grandes rasgos, el funcionamiento es el siguiente:

El usuario debe instalarse en su ordenar la suite que le permitirá ejecutar el editor que se encargará de la creación del contenido. Cuando tenga listo el contenido de la aplicación, el usuario podrá probar el código en el propio emulador que incluye la suite o directamente en un móvil Android generando un instalador.

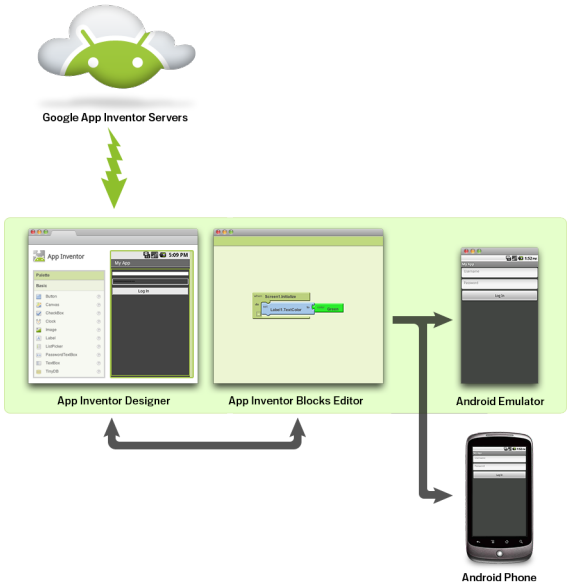


Figura 3: Mecanismo de Generación

Una vez se ha realizado la instalación, el usuario accederá a la parte generadora, conocida como "App Inventor Designer", donde se permite seleccionar los componentes que contendrá la aplicación y al "App Inventor Blocks Editor" donde se especifica el comportamiento (la funcionalidad) que tendrá un determinado componente.

Veamos estos dos bloques con más detalle: La Figura 4 muestra la interfaz de bloque "App Inventor Designer".

En la parte izquierda de la pantalla, están situados los controles que podremos usar en la aplicación (*Palette*), como por ejemplo botones, animaciones y en definitiva opciones propias de un móvil.

En la parte central podemos ver de manera esquemática, cómo irá quedando la aplicación (viewer), y en la parte derecha, las propiedades de los elementos que hayamos arrastrado al viewer y que tengamos seleccionado.

En concreto, en la Figura 4, se ha arrastrado el componente "Button" y el "Phone Call", con nombres "TopButton" y "TopCall" respectivamente, al viewer y a la derecha podemos ver las propiedades de éstos.

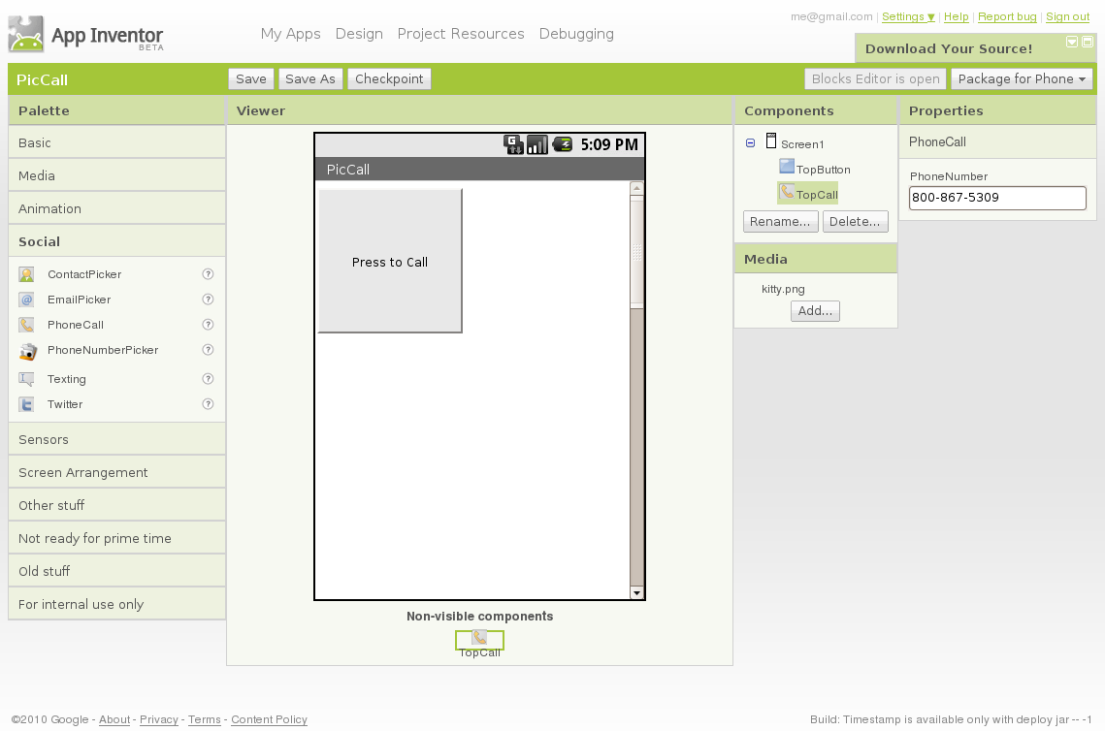


Figura 4: Designer

Suponiendo que el usuario ya ha configurado su pantalla, llega el momento de indicar cuál será la funcionalidad de los componentes que forman parte de ella.

Para ello, es necesario acceder al "App Inventor Blocks Editor, representado visualmente como piezas de un puzzle

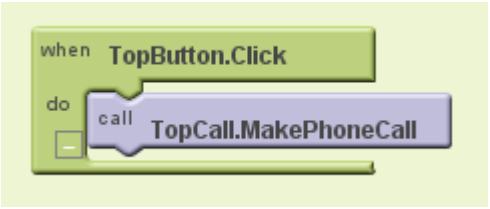


Figura 5: Blocks Editor

En este caso cuando (*when*) se haga click en el botón "TopButton", se llamará (*do*) al método "MakePhoneCall" del componente de la llamada "TopCall"

2.3. Buzztouch

Buzztouch[10] es una Aplicación Web gratuita que tiene como objetivo ayudar a construir aplicaciones CMS (Content Management Software) para iOS y Android.

Esta aplicación está pensada para todo tipo de usuarios, por su interfaz intuitiva, pero está más enfocada a usuarios programadores. Esto se debe a que una vez finaliza el proceso de

generación (como veremos a continuación) el usuario debe compilar el código con la SDK correspondiente a la plataforma. Este hecho resulta tremendamente útil para aquellos usuarios que quieran ver cómo se genera el código.

Buzztouch proporciona herramientas que permiten gestionar el contenido de las aplicaciones y su funcionamiento general es el siguiente:

El usuario se registra en la aplicación para poder acceder al sistema. Una vez realizado y validado el registro, buzztouch nos ofrece la posibilidad de crear una nueva aplicación

La pantalla que se muestra a continuación muestra dicha creación así como una pequeña guía sobre cómo se generará la nueva aplicación

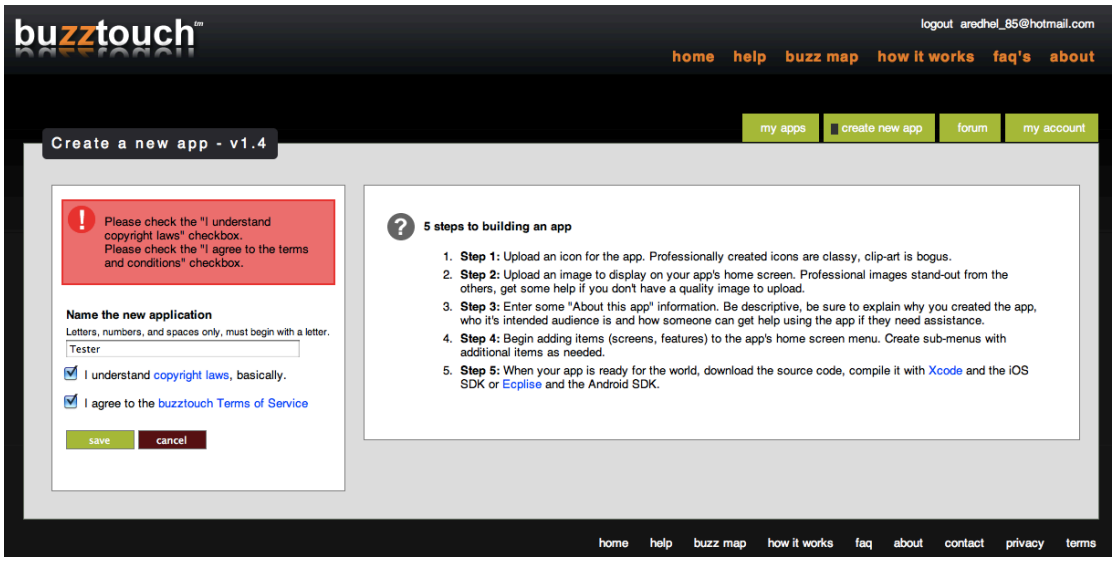


Figura 6: Creación de la aplicación

El siguiente paso es configurar las pantallas, menús, iconos, etc.. que aparecerán en la aplicación. Para ello buzztouch muestra un panel de control que permitirá realizar todas esas acciones.

En la parte izquierda del panel de control, nos encontramos con una vista previa que nos muestra que es lo que estamos creando.

En la parte central, tenemos la información de los elementos que van formando parte de nuestra aplicación, así como una lista desplegable que nos permite elegir cuál será el siguiente elemento que vamos a añadir.

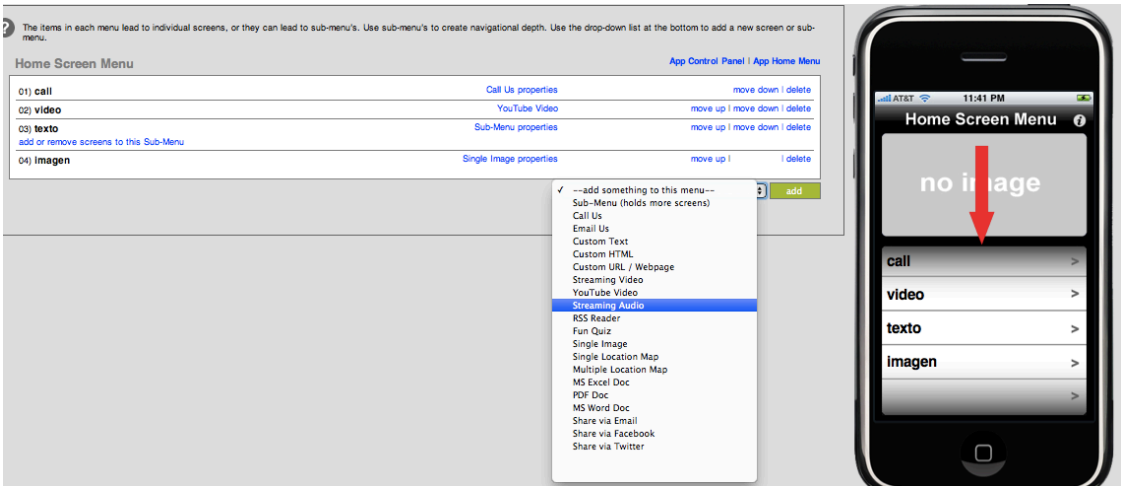


Figura 7: Panel de Control

Una vez hemos terminado de definir todo y de completar el contenido, hemos de descargar el código fuente desde la aplicación y compilarlo en base a la SDK correspondiente.

Después, la aplicación podrá subirse a la tienda de aplicaciones correspondiente y podrá ser descargada por cualquier usuario.

Figura 8: Generación

2.4. TEKORA

A diferencia de las aplicaciones mencionadas anteriormente, TEKORA [11] es una empresa cuyo objetivo principal es proporcionar a sus clientes una herramienta, que permita crear sitios web para móviles.

Estos sitios webs son compatibles con cualquier plataforma existente, puesto que sólo necesitan un navegador para funcionar.

Entre sus productos encontramos “Tekora Studio”:

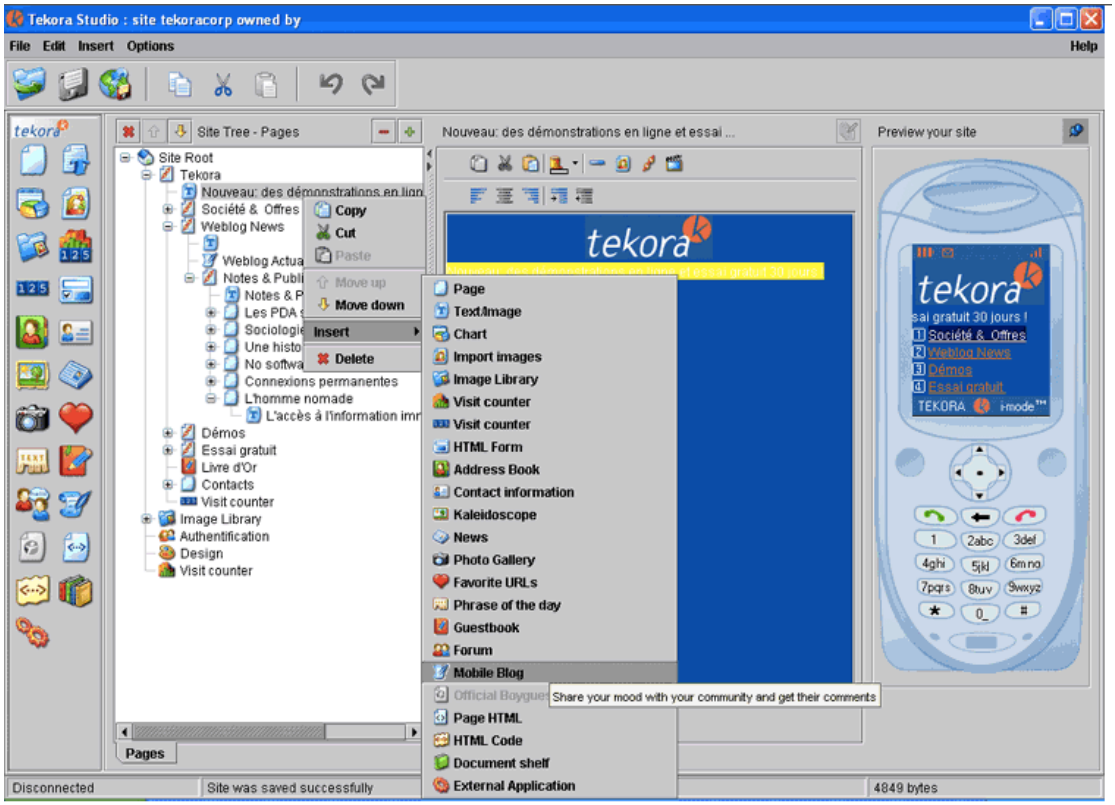


Figura 9: Interfaz Tekora Studio

La interfaz de Tekora Studio cuenta con los elementos típicos que ya hemos ido viendo en las demás aplicaciones. No es necesario tener conocimientos técnicos para crear la aplicación, ya que la interfaz que podemos ver, muestra de manera intuitiva los elementos que podemos incluir.

En la parte izquierda, podemos encontrar una botonera con los diferentes elementos que podemos ir añadiendo, como por ejemplo páginas, imágenes, contador de visitas, formularios, foros, gráficas, etc..

Junto a dicha botonera, tenemos un árbol desplegable que nos permite ver de un primer vistazo la estructura del sitio web, las páginas que va teniendo y los componentes que forman parte de las páginas.
A continuación, encontramos el editor de texto, que nos permite dar forma a los elementos que se han ido agregando y finalmente, un emulador a la parte derecha, para ver el resultado de la generación.

Así pues si quisiéramos empezar a construir el sitio web, crearíamos un nuevo nodo en el árbol y crearíamos un elemento de tipo texto, ya sea eligiéndolo desde la botonera o desde la opción "Insert", que aparece al efectuar clic sobre el elemento elegido con el botón secundario del ratón. Esta acción se encuentra visible en la Figura 9

Si quisiéramos otro elemento, por ejemplo un formulario, seguiríamos las mismas pautas. Puede verse el ejemplo ilustrativo en la Figura 10.

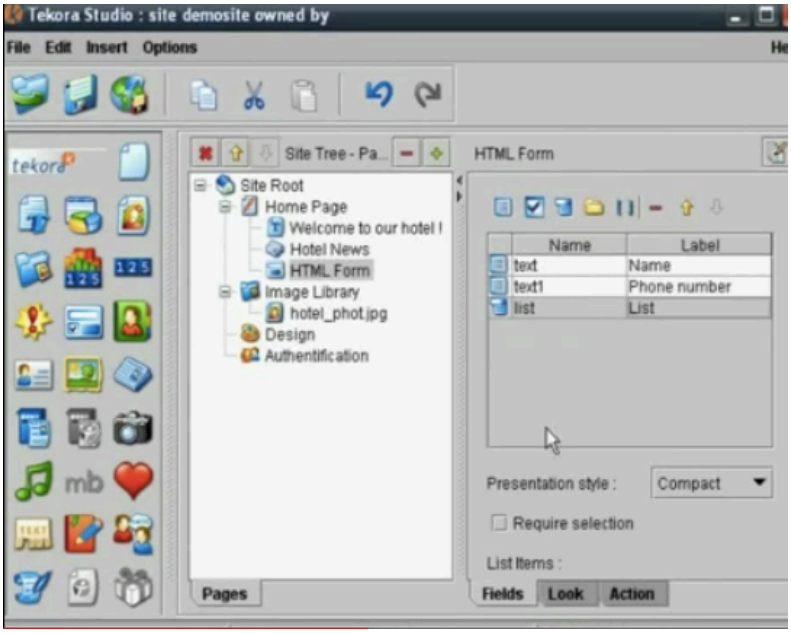


Figura 10: Creación de elemento formulario

El resultado de este proceso será publicado en los servidores de la empresa Tekora y accesible (como hemos mencionado anteriormente) desde cualquier tipo de móvil que pueda lanzar un navegador con acceso a la red

Finalizando con esta aplicación, mostramos un ejemplo de cómo quedaría una página web creada con Tekora Studio

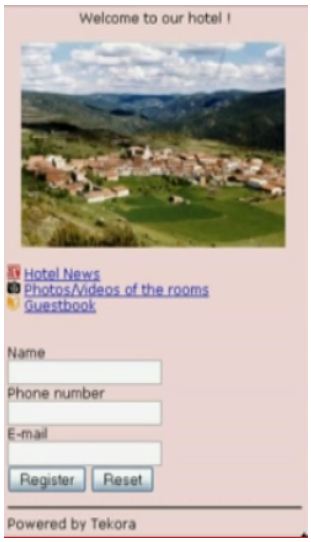


Figura 11: Página web creada con Tekora

2.5. PhoneGap

Por último, PhoneGap [12] también es distinto a todas a las aplicaciones que hemos visto anteriormente. Se trata de una plataforma abierta basada en HTML5, CSS y JavaScript.

Es capaz de generar aplicaciones móviles para Apple iOS, Android, Blackberry, WebOs, Windows Mobile y Symbian.

Puede considerarse como una creciente alternativa al desarrollo clásico de estos sistemas y su funcionamiento se basa en, crear aplicaciones con tecnologías web estándar y hacer una conversión para que pueda desplegarse en la plataforma deseada.

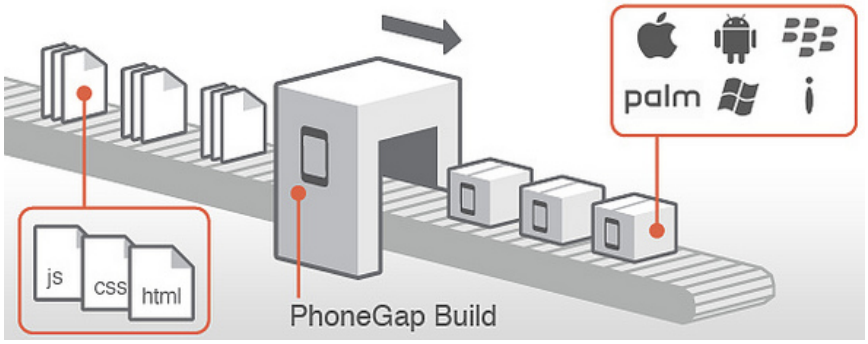


Figura 12: Mecanismo de Generación de PhoneGap

Una vez creada la aplicación, ésta se empaqueta con las librerías de PhoneGap de tal manera que permiten la distribución de ésta en las diferentes plataformas.
¿Cómo empezar a desarrollar? Según la información que proporciona la página web es necesario tener una distribución de la SDK de la plataforma que queremos crear (por ejemplo, si es para iPhone sería Xcode), la distribución estable más reciente de PhoneGap y crear un nuevo proyecto en la SDK importando los archivos que proporciona la distribución estable.

Realizando este procedimiento, sobre las distintas SDK disponibles, podemos obtener la misma aplicación lista para ejecutarse en las 6 plataformas disponibles.

2.6. Tabla Resumen

Mostramos un resumen que recoge las características principales de las soluciones que hemos visto y además, incluye las características de la solución propuesta MovilTurismo para una mejor comprensión del trabajo realizado.

Debido al gran número de soluciones encontradas, dividiremos en resumen en 2 tablas comparativas:

	MOVILEVENTO	APP INVENTOR	BUZZTOUCH
PLATAFORMA	**WEB**	**ESCRITORIO**	**WEB**
CONOCIMIENTOS NECESARIOS	**BÁSICOS**	**AVANZADOS**	**BÁSICOS**
FUNCIONALIDADES	**MENÚ**	**COMPONENTES MÓVILES (LLAMADAS, BOTONES, EVENTOS…)**	**MENÚ**
	AGENDA	**MULTIMEDIA**	**VIDEO**
	MAPA	**ANIMACIONES**	**LOCALIZACIÓN**
	CONTACTOS	**ACCESO REDES SOCIALES**	**RSS**
			HTML
			DOCUMENTOS (PDF, WORD..)
PLATAFORMAS QUE SOPORTA	**COMPATIBLES CON JAVA**	**ANDROID**	**IOS, ANDROID**
TIPO LICENCIA	**PAGO POR APLICACIÓN**	**OPENSOURCE**	**GRATUITA**

Tabla 1: Resumen de características 1

	TEKORA	PHONEGAP	MOVILTURISMO
PLATAFORMA	**ESCRITORIO**	**ESCRITORIO**	**WEB**
CONOCIMIENTOS NECESARIOS	**BÁSICOS**	**AVANZADOS**	**BÁSICOS**
FUNCIONALIDADES	**TODO TIPO DE COMPONENTES HTML (FORMULARIOS, BOTONES, MENÚ, GALERIA..)**	**COMPONENTES CREADOS CON HTML5, CSS Y JAVASCRIPT**	**MENÚ**
			IMÁGENES, TEXTO, ENLACES
			PERSONALIZACIÓN
			COMPONENTES HTML
PLATAFORMAS QUE SOPORTA	**CUALQUIERA CON NAVEGADOR WEB**	**IOS,IOS(XCODE4), ANDROID, BLACKBERRY, WEBOS, SYMBIAN**	**CUALQUIERA CON NAVEGADOR WEB COMPATIBLE CON HTML5**
TIPO LICENCIA	**PAGO**	**GRATUITA**	**GRATUITA**

Tabla 2: Resumen de características 2

2.7. Conclusiones

A lo largo de este apartado hemos repasado algunas aplicaciones, empresas y Frameworks que tratan de aportar su granito de arena a la generación de aplicaciones móviles.

Hemos visto que existe bastante variedad y que todas son una buena aproximación. Sin embargo es importante destacar algunas carencias o desventajas que destacan en todas ellas y que tal vez podrían mejorarse.

MovilEvento es una aplicación bastante completa, sin embargo un poco limitada en cuando a tipos de secciones. Únicamente deja usar 4 tipos y no permite crear ninguno más, ni personalizar los existentes.

App Inventor es una aplicación muy buena, pero resulta un poco compleja a la hora de establecer la funcionalidad de los componentes.. Otra cosa destacable es que no permite visualizar el código fuente ni subir la aplicación a Android Market, con lo que hoy por hoy, sirve para crear aplicaciones de "consumo propio".

Buzztouch es bastante intuitiva a nivel de usuario, pero tiene el problema que no genera la aplicación directamente, si no, que el usuario se descarga el código, lo compila y lo sube a la tienda de aplicaciones correspondiente, con el consiguiente retraso que supone depender de terceros para que publiquen la aplicación. Tampoco permite personalización, ni extensión de las características existentes.

Tekora parece sin embargo, un mecanismo bastante válido. Es una aplicación intuitiva, agradable a la vista y permite generar fácilmente los sitios webs para móviles. Es multiplataforma (se necesita sólo un navegador) y puede ejecutarse inmediatamente después de desplegarse en el servidor.

Por último, PhoneGap, supone una revelación y la mejor manera de generar una aplicación para móvil para cualquier plataforma. Tiene las mismas desventajas que Buzztouch, pero aún así, los resultado que se pueden llegar a obtener son visiblemente superiores. Este Framework está en constante crecimiento desde que vio la luz en 2008. Cuenta con una comunidad de usuarios bastante grande y los reviews y opiniones sobre este Framework son altamente positivos.

Destacar que ninguna de estas aplicaciones es una solución que use MDA.

En definitiva, es muy complicado aproximar la solución perfecta o la más eficiente para todos los tipos de usuario.

Los programas existentes servirán de base a los futuros desarrollos que están por llegar, ya que sin duda, esta es una tecnología en constante crecimiento y que será mejorada con los años para llegar a una solución estable, eficiente y sólida.

3. Contexto Tecnológico

En este capítulo se describen las tecnologías y los conceptos usados en el desarrollo de la presente tesina, así como las bases tecnológicas para realizarla.

Siendo este, el esquema tecnológico empleado en la solución:

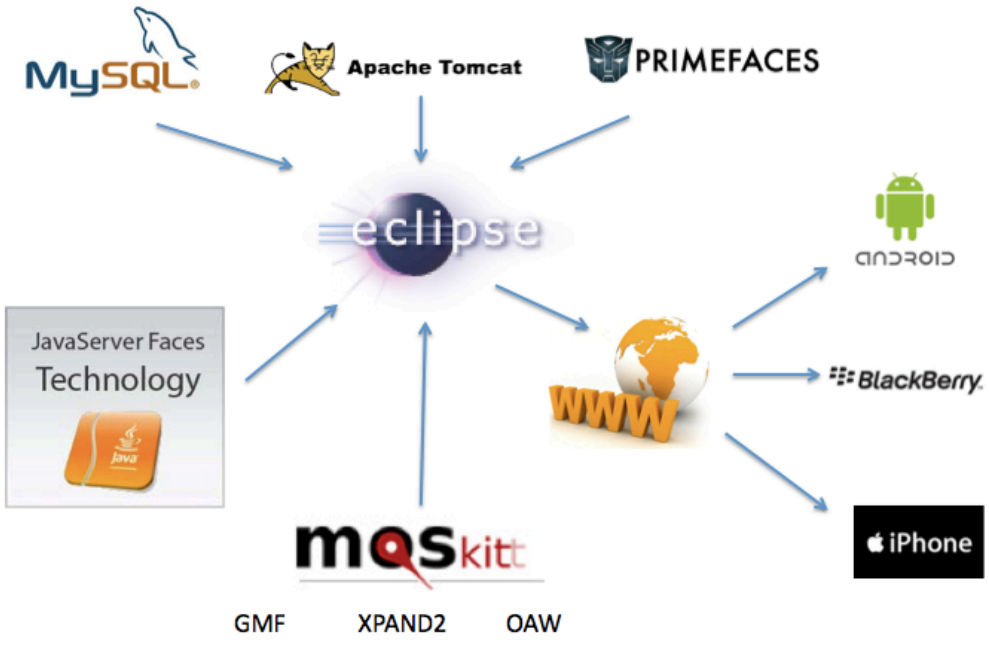


Figura 13: Esquema de la solución

En el punto 3.1 veremos un recorrido por la historia, evolución y estado actual de los dispositivos móviles. Siguiendo con el punto 3.2, analizaremos las bases que han asentado la creación de MovilTurismo. Por último, en el punto 3.3 veremos la tecnología implicada en el desarrollo, así como una descripción de sus características principales y funcionalidades.

3.1. Dispositivos Móviles

El primer servicio radiotelefónico, apareció en Estados Unidos a finales de 1940 y su utilidad residía en conectar realizar la conexión entre dispositivos móviles y vehículos conectados a una red pública.

En los años 60, un nuevo sistema lanzado por la empresa *Bell Systems,* conocido como IMTS (del inglés *Improved Mobile Telephone Service*), trajo gran cantidad de mejoras como marcación directa o un ancho de banda mayor [13]

El primer móvil basado en la tecnología IMTS fue desarrollado a finales de los años 60, principio de los años 70.

Desde entonces, la tecnología móvil ha ido mejorando sus capacidades y sus características hasta el día de hoy, donde el mercado de teléfonos inteligentes (smartphones) se incrementa anualmente.

Según *The Pew Research Center's Internet & American Life Project* [14] el 35% de los adultos americanos, tiene su propio smartphone y más allá, el 68% de éstos, usa su dispositivo para acceder a internet diariamente, teniendo en cuenta que un 25% usa el smartphone como su dispositivo principal para acceder a internet.

Con estos dispositivos claramente encauzados para convertirse en un dispositivo ubicuo, es interesante pararse a pensar en el impacto que esto supone en las pequeñas empresas, especialmente, en aquellas que ofrecen sus servicios a través de la Web.

¿Qué impacto tienen en ellas? ¿Qué pueden hacer al respecto? Ellos recomiendan dejar de crear variaciones "móviles" del sitio web empresarial (que normalmente se traduce en el sitio web original con funcionalidad limitada y con funcionamiento incorrecto, o no tan correcto como debiera ser) y centrarse en hacer el sitio web más navegable desde el punto de vista de las pantallas táctiles y el pequeño tamaño de los smartphones actuales.

Un ejemplo de esto, sería evitar las presentaciones Flash pesadas y los complicados menús desplegables así como ofrecer una interfaz limpia que se adapte a la perfección a las pantallas de los móviles Android, iPhone, iPads y equipos de escritorio de Windows 7.

Otro punto a tener en cuenta, es si la empresa implicada está interesada en crear una aplicación móvil para permitir a los usuarios acceder a su negocio en línea. Para ello, debe considerar el coste del desarrollo y si la aplicación puede proporcionar funcionalidades que no son disponibles vía la web principal (como por ejemplo reconocer la localización del usuario o integrarse con otras aplicaciones) para poder sopesar si realmente vale la pena. Si no ofrece funcionalidades adicionales, una alternativa podría ser emplear el coste del

desarrollo en mejorar la web principal tal y como se ha descrito anteriormente.

Aunque también es cierto, que hay un gran incremento en la tendencia de tener una versión móvil de cualquier sitio web debido a (como hemos mencionado anteriormente) el incremento del número de smartphones en el mercado, con lo que supone un mayor uso de estos dispositivos.

Es por ello, que ninguna empresa o marca, desea perder terreno en este campo, puesto que en 2010 se descargaron 5 mil millones de aplicaciones y algunas predicciones sitúan este valor en 21 mil millones de aplicaciones de cara al año 2013.

Veamos algunos números según ShoutEm [15]

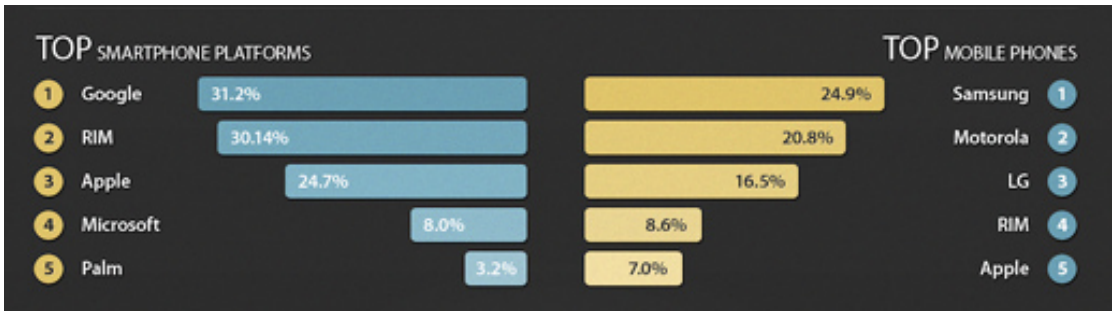


Figura 14: Estadísticas

Los datos nos muestran que hoy por hoy, la plataforma más usada es la plataforma de Google (Android), seguida de cerca por RIM (Blackberry) y por Apple (iOS). En cuando al modelo de smartphone más demandado, se sitúa a la cabeza Samsung

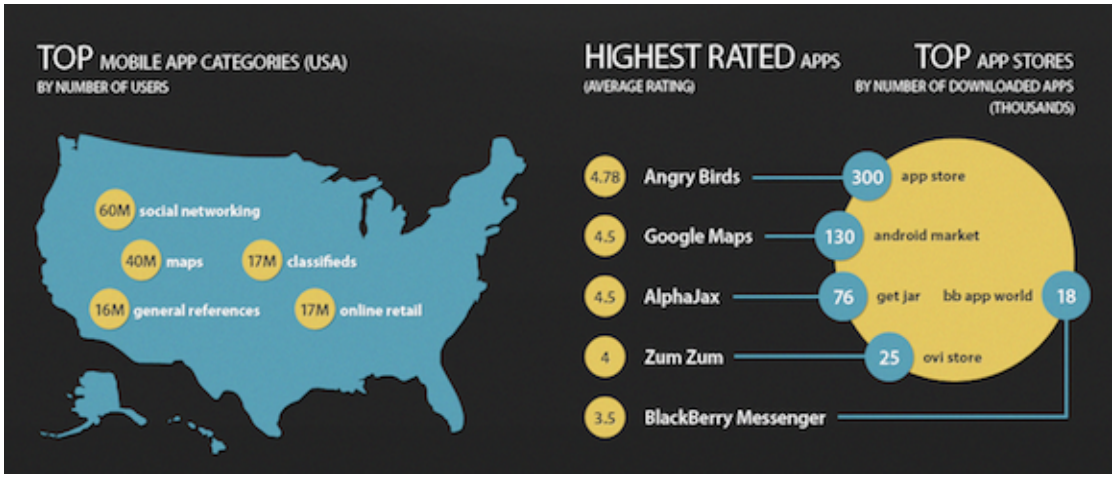


Figura 15: Estadísticas 2

Observemos el ranking de descargas de las aplicaciones en cada tienda, siendo por ejemplo "Angry Birds", la aplicación más descargada en la *App Store*, "Google Maps" en el *Android Market* o "BlackBerry Messenger" para *Bb app world*.

Para finalizar vemos que en cuanto a la clasificación de tipos de aplicaciones, reciben un mayor número de usuarios aquellas que tienen que ver con las redes sociales y con la localización y mapas, siendo ésta, la predicción para 2012.

Figura 16: Predicción 2012

3.2. Tecnología Empleada

3.2.1. JSF (JavaServer Faces)

JavaServer Faces (JSF) [17] es el Framework para aplicaciones Web en Java de Sun Microsystems, liberado en Marzo del 2004 con el objetivo de desarrollar aplicaciones web de forma parecida a como se construyen aplicaciones con Java Swing, AWT, SWT o similares

Está orientado hacia la interfaz gráfica de usuario (GUI), facilitando así su desarrollo y proporcionando una separación entre comportamiento y presentación, además de proporcionar su propio servlet como controlador, implementando así los principios del patrón MVC (modelo – vista – controlador) y dando como resultado un desarrollo más simple y una aplicación mejor estructurada.

Tradicionalmente, las aplicaciones Web se han codificado mediante páginas JSP (Java Server Pages) que recibían peticiones a través de formularios y construían como respuesta páginas HTML mediante la ejecución (directa o indirecta) de código Java.

Con la tecnología JSF, la interfaz de usuario se ejecuta en el servidor y se renderiza en el cliente

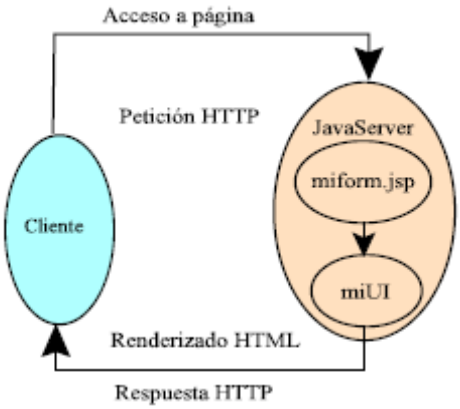


Figura 17: Esquema JSF

Así, se pretende facilitar la construcción de estas aplicaciones proporcionando un entorno de trabajo (Framework), que gestiona las acciones producidas por el usuario en su página HTML y las traduce a eventos, que son enviados al servidor, con el objetivo de reflejar los cambios pertinentes provocados por dichas acciones.

Otra importante característica es que, a pesar de que HTML es su lenguaje de marcas por defecto, no está limitado a éste (ni a ningún otro), ya que JSF tiene la capacidad de usar diferentes tecnologías de presentación, como por ejemplo JSP, Javascript o XHTML (JSF 2.0).

3.2.1.1. Ciclo de Vida

El ciclo de vida de JSF es similar al de una página JSP: el cliente hace una petición HTTP y el servidor responde con la página en HTML.

Sin embargo, debido a las características que ofrece JSF, su ciclo de vida incluye algunos pasos más:

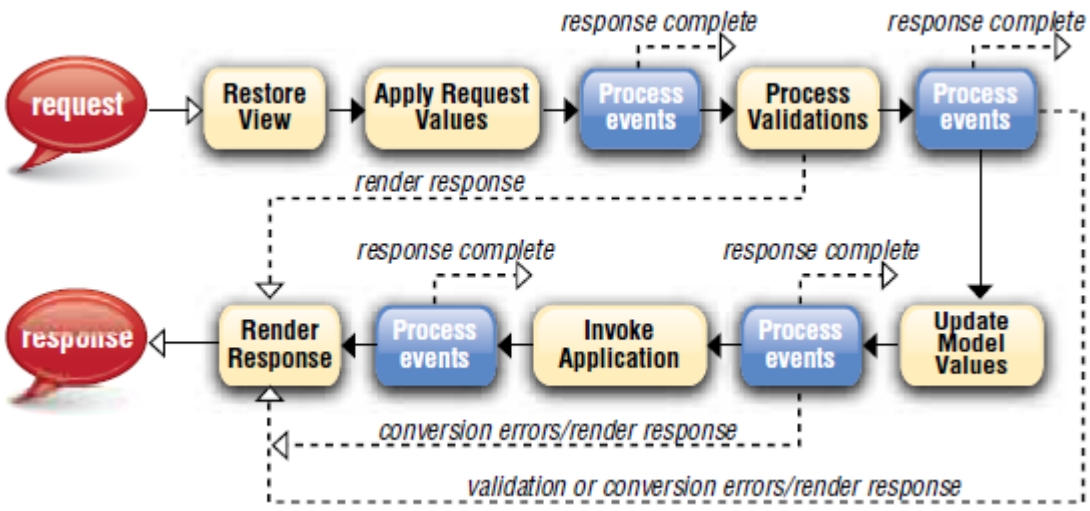


Figura 18: Ciclo de vida

El proceso de una petición estándar incluye 6 fases (en color claro):

- *Restore View*
- *Apply Request Values*
- *Process Validation*
- *Render Response*
- *Invoke Application*
- *Update Model Values*

Los elementos "Process Events" (en azul) representan la ejecución de cualquier evento producido durante el ciclo de vida.

Describimos a continuación el funcionamiento de cada fase:

Restore view: Es la etapa que se inicia cuando se realiza una petición. Su objetivo es la creación de un árbol con todos los componentes de la página de la que proviene la petición

Apply Request Values: obtiene el valor correspondiente a la petición realizada de cada uno de los componentes del árbol creado en la fase anterior

Process Validations: Se validan los valores recogidos para cada componente del árbol según las reglas que se hayan declarado

Update Model Values: Los valores locales de los componentes se usan para actualizar los beans a los que están ligados. Eso se llevará a cabo sólo si las validaciones de la fase anterior tuvieron éxito.

Invoke Application: Se ejecuta la acción correspondiente al evento inicial que dio comienzo a todo el proceso.

Render Response: La respuesta se renderiza y se devuelve al cliente

Dependiendo del éxito o fracaso de las tareas en cada una de las fases del ciclo de vida, el flujo normal descrito puede cambiar hacia caminos alternos

3.2.1.2. Estructura de una aplicación JSF

Típicamente, para lanzar aplicaciones JSF se necesita una implementación de su especificación (librerías) y un contenedor Web para Java.

Los elementos más comunes que forman este tipo de aplicación son:

- Archivos **JSP**: constituyen la interfaz gráfica de la aplicación y que contienen las diversas funcionalidades de JSF, como los tags que representan los componentes GUI. (en la versión 2.0 de JSF se substituyen por XHTML)

- Archivos **XML**: como por ejemplo el archivo faces-config.xml, que almacena las diferentes configuraciones y los elementos (beans) a utilizar en la aplicación

- Archivos **JAVA**: Hacen la función de un Bean (A parte de su función habitual)

- Archivos **bundle**: Para paquetes de mensajes (internacionalización)

- Archivos **XHTML**: Sustituyen a los archivos JSP a partir de la versión 2

En la siguiente imagen se ilustra la estructura anteriormente mencionada

Figura 19: Estructura aplicación JSF

Un ejemplo de archivo JSP sería

```
<html>
   <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
   <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
   <f:view>
      <head>
         <link href="styles.css" rel="stylesheet" type="text/css"/>
         <f:loadBundle basename="com.corejsf.messages" var="msgs"/>
         <title>
            <h:outputText value="#{msgs.tituloVentana}"/>
         </title>
      </head>

      <body>
         <h:outputText value="#{msgs.tituloPagina}" styleClass="emphasis"/>
         <p/>       
         <h:form>
            <h:panelGrid columns="3">
               <h:outputText value="#{msgs.nombre}"/>
               <h:inputText id="nombre" value="#{form.nombre}" required="true" >
               </h:inputText>
               <h:message for="nombre" />

               <h:outputText value="#{msgs.apat}"/>
               <h:inputText id="apat" value="#{form.apat}" required="true" >
               </h:inputText>
               <h:message for="apat" />

               <h:outputText value="#{msgs.amat}"/>
               <h:inputText id="amat" value="#{form.amat}" required="true" >
               </h:inputText>
               <h:message for="amat" />
```

Figura 20: JSP

y un ejemplo de archivo faces-config.xml

```
<?xml version="1.0"?>

<!DOCTYPE faces-config PUBLIC
   "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
   "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>

      <managed-bean>
            <managed-bean-name>form</managed-bean-name>
            <managed-bean-class>com.corejsf.RegisterForm</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>

      <navigation-rule>
            <from-view-id>/index.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>exito</from-outcome>
                  <to-view-id>/resultados.jsp</to-view-id>
            </navigation-case>
      </navigation-rule>

</faces-config>
```

Figura 21: faces-config.xml

3.2.1.3. JavaBeans

De acuerdo a la especificación, los JavaBeans se definen como “componentes de software reutilizables” que pueden manipularse desde una herramienta de desarrollo.

Un JavaBean se pude ver como una clase Java tradicional que sigue ciertas especificaciones de programación

Las características más importantes de un bean, son las propiedades que ofrece. Por ejemplo, cualquier atributo de un bean cuenta con:

- Nombre
- Tipo
- Getters / Setters (métodos para escribir o leer el valor del atributo)

A continuación vemos un ejemplo ilustrativo en el que se puede ves un JavaBean con 2 atributos y sus correspondientes métodos get y set para devolver/establecer el valor del atributo nombre y del atributo apellido

```
package com.jsf;
public class BeanUsuario {
   private String nombre;
   private String apellido;

   // PROPERTY: nombre
   public String getNombre() { return nombre; }
   public void setNombre(String nuevoValor) { nombre = nuevoValor; }

   // PROPERTY: apellido
   public String getApellido() { return apellido; }
   public void setApellido(String nuevoValor){ apellido = nuevoValor; }
}
```

Figura 22: Ejemplo JavaBean

En JSF los beans tienen la función de conectar las clases Java con las páginas web, es decir, de conectar la interfaz gráfica con la lógica aplicativa.

Para poder hacer uso de un bean, debe declararse en el fichero faces-config.xml de la siguiente manera

```
<managed-bean>
   <managed-bean-name>usuario</managed-bean-name>
   <managed-bean-class>com.jsf.BeanUsuario</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Figura 23:faces-config.xml

Donde:

managed-bean-name: Indica el nombre del objeto
managed-bean-class: Indica la clase del objeto
managed-bean-scope: Indica el alcance del objeto

Así pues, podría traducirse como: "*Construye un objeto de la clase com.jsf.BeanUsuario, llámalo usuario y mantenlo activo mientras dure la sesión*"

Una vez declarado el bean, éste puede ser usado por los componentes de JSF a través de value binding expressions, cuya sintaxis es la siguiente:

<h: inputText id= nombre value="#{NombreBean.NombreAtributo}"/>

De esta manera, se logra mapear el valor del componente en el atributo del bean, pudiéndose utilizar el valor tanto para las diferentes operaciones que deban realizarse con él como para mostrarlo por pantalla.

3.2.1.4. Navegación entre páginas

En JSF la navegación [18] entre páginas de la aplicación vendrá determinada por las reglas declaradas en el fichero faces-config.xml.

Estas reglas serán evaluadas con la respuesta que devuelva el evento que se genere en la página desde la que queremos navegar.

Cuando el usuario hace clic en un botón se genera un evento *submit*. En este punto, la aplicación debe analizar los datos que se han enviado y determinar cuál será la siguiente página a mostrar.

```
<h: commandButton  value=”#{NombreBean.NombreMétodo}” action =”#{NombreBean.NombreMétodo}/>
```

Para determinar la vista a mostrar, la aplicación toma la respuesta generada por el atributo *action* del componente que originó el evento, y en base a ésta y a las reglas establecidas, selecciona la página adecuada.

3.2.1.5. Manejo de Eventos

Los eventos juegan un papel muy importante dentro de JavaServer Faces. Los componentes GUI de JSF responden a las acciones del usuario disparando eventos, que serán controlados por código de la aplicación. Se conocen como *listeners*.

JSF soporta dos tipos de eventos para los componentes GUI:

- Value Change
- Action

La idea central del manejo de eventos es registrar los *listeners* adecuados para cada componente, de tal manera que se realicen las operaciones apropiadas y deseadas.

Eventos *Value Change*

Estos eventos son generados por componentes de entrada, tales como *h:inputText, h:selectManyMenu* y *h:selectOneRadio*, cuando su valor cambia y se realiza un submit.

Para agregar un listener de este tipo, además de asociarlo con el correspondiente componente GUI y a través de un *value binding expression* indicar el método que se invocará al dispararse el evento, debe añadirse el atributo *onchange* para forzar que se haga el submit:

```
<h:selectOneMenu value="#{bean.pais }" onchange="submit()" valueChangeListener="#{bean.cambioPais}">
<f:selectItems value="#{bean.nombresPaises }"/>
</h:selectOneMenu>
```

Eventos *Action*

Estos eventos son lanzados por componentes tipo *command*, tales como *h:commandButton* y

h:commandLink, cuando el botón o el link son presionados.

A diferencia de los eventos Value Change, los eventos Action realizan el *submit* de manera automática, por lo cual sólo debe agregarse el atributo *actionListener* y la referencia al método a invocar, al componente GUI deseado, sin necesidad del atributo *onchange*:

<h:commandLink actionListener="#{bean.linkUsado}"> </h:commandLink>

Así mismo, el método linkUsado debe tener como parámetro una variable de tipo *ActionEvent*.

3.2.1.6. Conversión y Validación

Antes de que los datos introducidos por el usuario sean almacenados en los beans, éstos deben pasar por dos procesos necesarios: conversión y validación, siempre en este orden.

La conversión es el proceso a través del cual se transforman los datos ingresados por el usuario en los tipos de datos que mapean en sus correspondientes variables de su bean.
Es decir, los datos que proporciona el usuario, tratados como cadenas de texto, se transforman en *int*, *Date* o el tipo de dato que se ha definido para cada uno de ellos en el bean especificado.

La validación es el proceso mediante el cual JSF analiza los datos del usuario de manera semántica, de acuerdo a ciertas reglas especificadas para cada uno de estos, como pueden ser la longitud mínima y máxima para un campo.

Ambos procesos tienen como objetivo filtrar los datos del usuario, de tal manera que los valores que se guardan en los beans sean coherentes y aceptados por el proceso llevado en la lógica aplicativa.

Conversión estándar

JavaServer Faces proporciona algunos convertidores para los tipos de datos básicos. Por ello, puede realizar la conversión de manera automática para los tipos de datos *Boolean, Byte, Character, Double, Float, Integer, Long, Short, BigDecimal, BigInteger*, y para sus tipos primitivos.

También permite realizar la conversión a números, independientemente del tipo, pudiendo darles el formato deseado, como cantidad de dígitos en la parte entera o decimal.

Para lograr esto, lo único que debe hacerse es añadir al componente GUI el convertidor adecuado con sus diferentes atributos.

Por ejemplo: se usa el convertidor *f:convertNumber* y su atributo *minFractionDigits* para

determinar el número mínimo de dígitos en la parte decimal:

```
<h:inputText value="#{bean.cantidad}">
<f:convertNumber minFractionDigits="2"/>
</h:inputText>
```

Validación estándar

JSF proporciona validación para la longitud de cadenas de texto y para rangos de números.

```
<h:inputText value="#{bean.direccion}">
<f:validateLength minimum="13"/>
</h:inputText>
```

Así mismo, permite realizar validación para campos que son necesarios o requeridos, añadiendo únicamente el atributo required al componente GUI deseado:

```
<h:inputText id=nombre value="#{bean.nombre}" required="true"/>
```

3.2.2. Primefaces

Primefaces [19] es una librería de componentes visuales "*open source*" para JSF, desarrollada y mantenida por la compañía Turca "*Prime Technology"*

Sus principales características son:

- Un conjunto de más de 100 componentes (HTMLEditor, AutoComplete, Charts, etc…)
- soporte nativo de Ajax, incluyendo Push/Comet
- Kit para crear aplicaciones web para móviles
- Un jar, cero configuraciones y no necesita dependencias
- 30 temas prediseñados
- Documentación extensiva

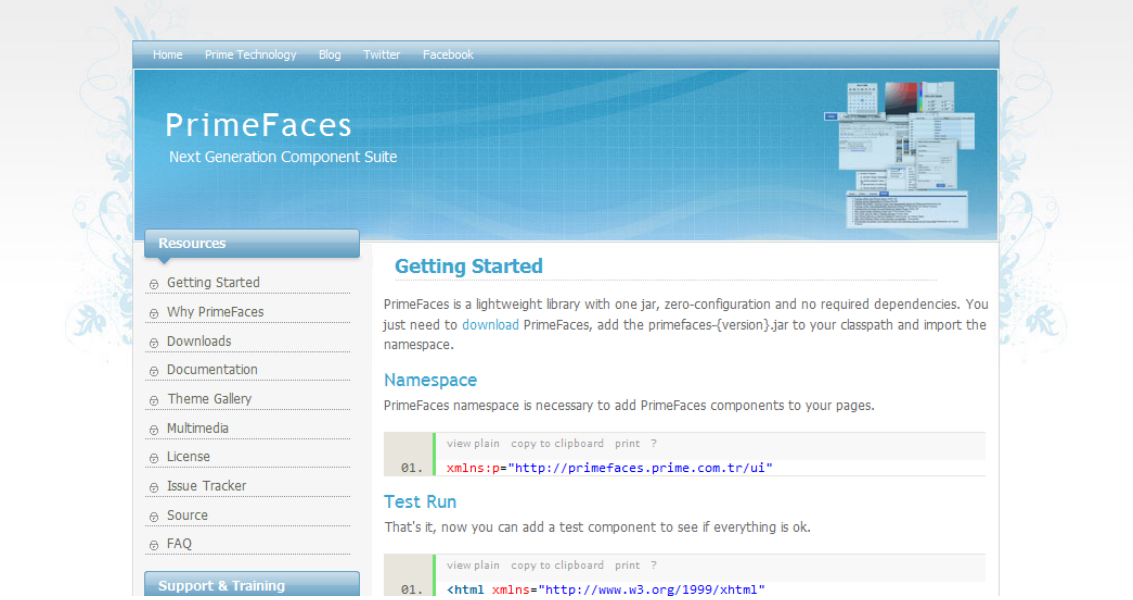


Figura 24: Página principal PrimeFaces

La web de Primefaces nos ofrece información acerca de cómo usar la librería, así como un apartado "*Showcase*" dónde ver cómo funcionan los componentes y como hay que definirlos

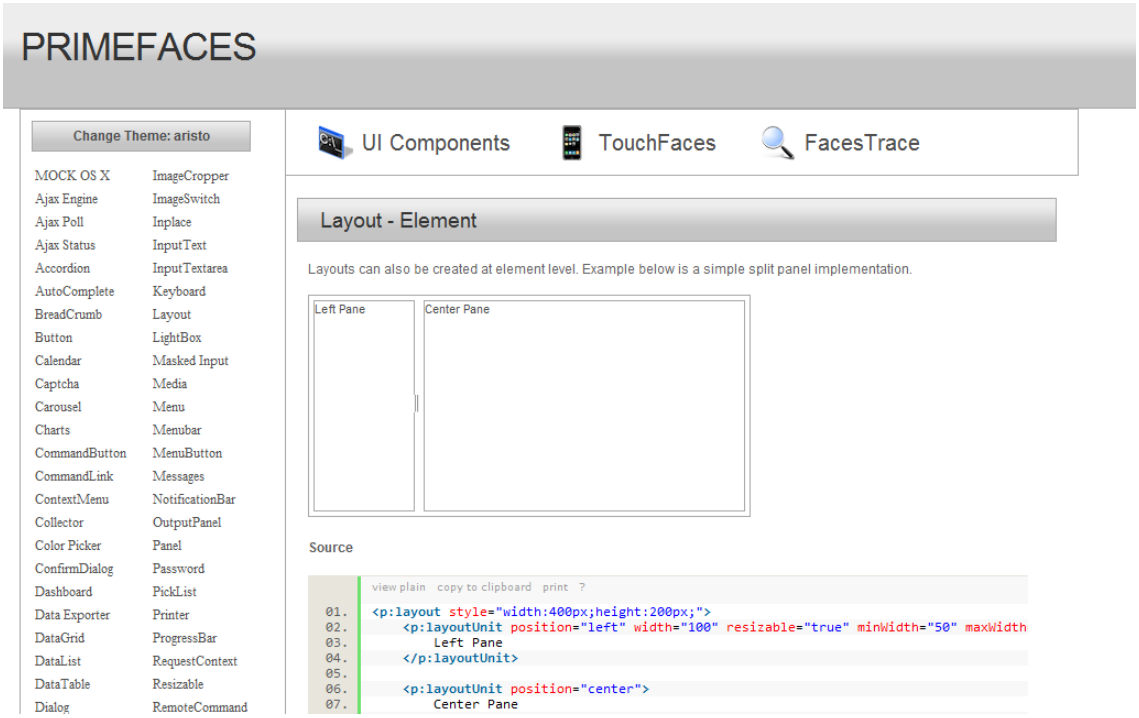


Figura 25: Showcase

3.2.2.1. Instalación

Primefaces [20] consta de un único jar llamado *primefaces-{version}.jar*, donde version es la versión actual de la distribución de primefaces, actualmente la 2.2.1

Existen dos maneras de descargar el jar para su uso: descargándolo manualmente o si el usuario que lo descarga, es un usuario de Maven, puede especificar la librería como una dependencia.

Descarga Manual

Acceder directamente al apartado "Downloads" de la web de primefaces y descargar la distribución correspondiente:

```
http://www.primefaces.org/downloads.html
```

Descarga con Maven

Establecer como groupId de la dependencia *org.primefaces* y como *artifactId primefaces*

```
<dependency>
      <groupId>org.primefaces</groupId>
      <artifactId>primefaces</artifactId>
      <version>2.2</version>
</dependency>
```

Adicionalmente a esta configuración, es necesario añadir el repositorio de maven de *Prime Technology* a la lista de repositorios que maven puede descargar

```
<repository>
      <id>prime-repo</id>
      <name>Prime Technology Maven Repository</name>
      <url>http://repository.prime.com.tr</url>
      <layout>default</layout>
</repository>
```

3.2.2.2. Dependencias

PrimeFaces solo necesita una versión superior a la versión 5 de JAVA y una implementación JSF 2.0 (o inferior).

Existen además, algunas librerías opcionales para ciertas características.

Dependency	Version	Type	Description
JSF runtime	2.0+	Required	Apache MyFaces or Oracle Mojarra
itext	2.1.7	Optional	PDF export support
apache poi	3.2-FINAL	Optional	Excel export support
Commons-fileupload	1.2.1	Optional	FileUpload
commons-io	1.4	Optional	FileUpload
atmosphere-runtime	0.5.1	Optional	Ajax Push
atmosphere-compat	0.5.1	Optional	Ajax Push

Tabla 3.2.2.2: Dependencias

3.2.2.3. HelloWorld

Una vez descargado el jar y añadido al classpath, hay que añadir el namespace de Primefaces al inicio de la página para poder usar los componentes.

A continuación vemos el ejemplo de una página simple:

```
<html xmlns="http://www.w3c.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.prime.com.tr/ui">

<h:head>
</h:head>

<h:body>

      <p:spinner />

</h:body>

</html>
```

3.2.3. Otras tecnologías

Mostramos a continuación un breve resumen del resto de tecnologías empleadas

3.2.3.1. Eclipse (Helios)

En la web oficial de Eclipse [21] se define como "*An IDE for everything and nothing in particular*" (un IDE para todo y para nada en particular).

Eclipse es, en el fondo, únicamente un armazón (workbench) sobre el que se pueden montar herramientas de desarrollo para cualquier lenguaje, mediante la implementación de los plugins adecuados.

La arquitectura de plugins de Eclipse permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, etc.

El IDE Eclipse es una de las herramientas que se engloban bajo el denominado *Proyecto Eclipse*. El Proyecto Eclipse aúna tanto el desarrollo del IDE Eclipse como de algunos de los plugins mas importantes (como el JDT, plugin para el lenguaje Java, o el CDT, plugin para el lenguaje C/C++).

Este proyecto también alcanza a las librerías que sirven como base para la construcción del IDE Eclipse (pero pueden ser utilizadas de forma completamente independiente), como por ejemplo, la librería de widgets SWT.

En su origen, era un proyecto de desarrollo OpenSource, soportado y mantenido en su totalidad por IBM.

Bajo la dirección de esta compañía, se fundó el *Consorcio Eclipse* al cual se unieron algunas empresas importantes como Rational, HP o Borland.

Desde el día 2 de febrero de 2004, el Consorcio Eclipse es independiente de IBM y entre otras, está formado por las empresas: HP, QNX, IBM, Intel, SAP, Fujitsu, Hitachi, Novell, Oracle, Palm, Ericsson y RedHat, además de algunas universidades e institutos tecnológicos.

3.2.3.2. Apache Tomcat 7.0

Tomcat [22]es un servidor Web con soporte de servlets y JSPs. No se trata de un servidor de aplicaciones, como JBoss o JOnAS y su motor de servlets se presenta a menudo en combinación con el servidor web Apache

Puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

3.2.3.3. MySQL

MySQL [23] es una base de datos open source muy popular y es un sistema de administración de bases de datos (Database Management System, DBMS) para bases de datos relacionales.

Existen muchos tipos de bases de datos, desde un simple archivo hasta sistemas relacionales orientados a objetos. MySQL, como base de datos relacional, utiliza múltiples tablas para almacenar y organizar la información.

Fue escrito en C y C++ y destaca por su gran adaptación a diferentes entornos de desarrollo, permitiendo su interactuación con los lenguajes de programación más utilizados como PHP, Perl y Java y su integración en distintos sistemas operativos.

También es muy destacable, la condición de open source de MySQL, que hace que su utilización sea gratuita e incluso se pueda modificar con total libertad, pudiendo descargar su código fuente.

3.3. Lenguaje de especificación: el metamodelo

La creación de aplicaciones siguiendo MDA, en este caso en concreto, la de aplicaciones Web para móviles, no sería posible sin el metamodelo *ecore*.

Pero antes de comentar el metamodelo y para tener una visión más general, vamos a ver algunas definiciones:

Un modelo, [28] se conoce por ser una representación abstracta de un dominio en concreto. Dicho modelo, captura los elementos importantes y las relaciones entre ellos. Por otro lado, un metamodelo describe las características de los modelos en sí y de los elementos que lo componen.

Por ejemplo, cualquier modelo UML contiene:

- Clases
- Atributos
- Operaciones
- Relaciones
- Herencia
- …

El *ecore* ha sido el metamodelo empleado para crear nuestra plataforma. Dicho metamodelo, especifica las características de las clases (EClass), sus características estructurales (EStructuralFeatures) , atributos, operaciones y herencia.

Está orientado a Java, ya que permite la especificación de tipos de datos básicos y esta imagen nos muestra su aspecto:

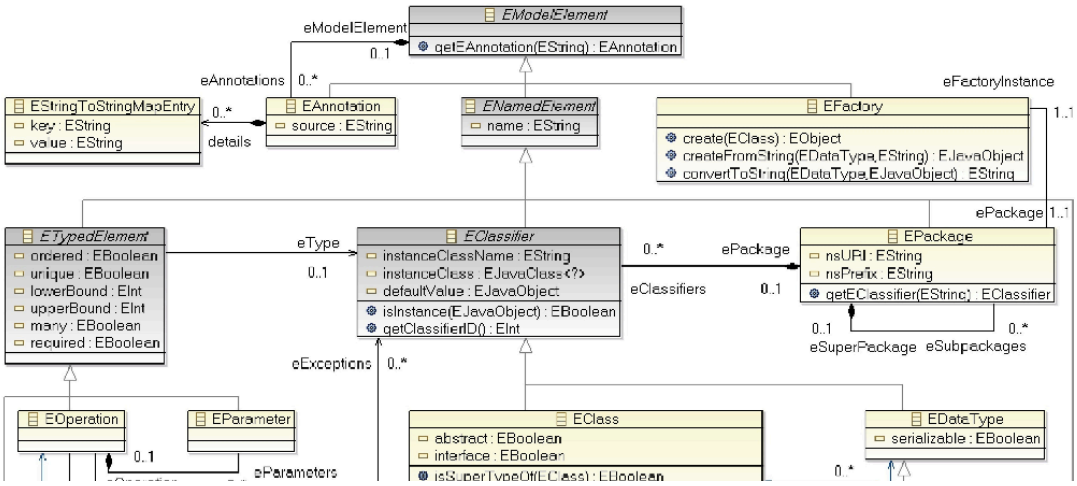


Figura 26: Ejemplo de ecore

Un metamodelo, puede serializarse a XMI (conservando su condición de modelo) y esta característica ha sido esencial en nuestro desarrollo. Este sería un ejemplo de serializado XMI de la instancia Tree.

```
<tree:Node xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:tree="http://www.example.org/tree"
label="root">
<children label="A"> <children label="X"/>
</children> <children label="B">
<children label="Y"/> </children>
</tree:Node>
```

Para la realización de MovilTurismo, se ha partido de un metamodelo dado, en el que se recogen las principales elementos necesarios para abordar la generación de la plataforma. Dado lo extenso del diagrama, es posible verlo en su totalidad en el Anexo 2, del presente documento.

4. La plataforma MovilTurismo

A lo largo de este punto, vamos a ver en profundidad el análisis de la plataforma.

Para ello, en el punto 4.1 mostraremos un visión general y veremos cómo se pretende desarrollar aplicaciones Web para móviles siguiendo las directrices de MDA. En el punto 4.2 mostraremos el pilar fundamental sobre el que se asienta la generación por modelos: el metamodelo. Por otro lado en el punto 4.3, hablaremos sobre la interfaz de usuario y su interacción con éste y en el punto 4.4, los requisitos esperados que se deben cumplir. Por último en el punto 4.5 mostraremos los diagramas propios de un análisis software.

4.1. Visión general

En el capítulo 2 del presente documento, hemos realizado un recorrido por algunas de las soluciones existentes para generar automáticamente código. Cada una de estas soluciones intenta abordar la solución de la mejor manera posible. En nuestro caso, se ha tratado de coger lo mejor o más ventajoso de cada una y de aprovecharlo.

Como hemos comentado anteriormente MovilTurismo es una plataforma Web concebida para permitir a los usuarios finales crear aplicaciones móviles a su gusto sin necesidad de conocimientos previos de programación.

Para conseguirlo, hemos acudido a MDA en pos de aprovechar las ventajas que nos ofrece. MDA nos brinda la posibilidad de definir nuestro sistema software en base a la representación abstracta de un dominio, en otras palabras, en base a un modelo.

La idea principal es tomar esa representación abstracta que se corresponde con el dominio que queremos emplear (en nuestro caso, inicialmente las guías turísticas, pero ha ido evolucionando hacia cualquiera) y extraer de ella los elementos importantes y las relaciones entre ellos

La extracción de esta información nos permite emplearla para establecer las bases de una aplicación, que relación existen entre sus componentes, cómo puede aprovecharse y qué información podemos usar, etc...

Para ello, se ha empleado la tecnología expuesta en el anterior punto, tratando de abordar este cometido de la mejor manera posible.

El objetivo ha sido claro en todo momento, y es, crear una herramienta que aproveche este funcionamiento, que lo moldee y que lo adapte convenientemente para que sea capaz de generar aplicaciones Web para móviles.

Hemos podido comprobar gracias al estudio previo, que la multiplataforma, es una característica muy perseguida en las aplicaciones existentes. Por ello, hemos querido dar la posibilidad de que el usuario pueda elegir para qué plataforma desea crear su aplicación.

Dispone de 2 opciones: por un lado la generación para la plataforma *BlackBerry* (nativa) y por otro lado, la posibilidad de generar para *HTML5* lo que le permite ejecutarse en cualquier plataforma que sea capar de ejecutar este lenguaje.

Cuando se realiza la generación en base a la plataforma *BlackBerry*, se usa una solución similar a la solución **BuzzTouch**. Es decir, se genera el código y luego debe compilarse en la suite correspondiente para, si se desea, subirla finalmente a la tienda de aplicaciones.

Sin embargo cuando se hace en base a cualquier plataforma, se usa la solución que emplea **TEKORA**. Esto es, generar una página Web (en nuestro caso con *HTML5*) y que se pueda abrir desde cualquier dispositivo móvil con un navegador Web.

A pesar de esta distinción, la aplicación será, en ambos casos, muy parecida estética y funcionalmente.

Para una mayor comprensión la especificación de casos de uso se encuentra en el anexo 1 del presente documento.

4.2. Lenguaje para la Especificación de Aplicaciones Web para móviles

En el presente apartado, vamos a comentar el metamodelo que se ha empleado en la creación de la plataforma. Debido al gran tamaño del diagrama vamos a verlo por partes, aunque puede consultarse entero en el Anexo 2 del presente documento.

La entidad principal del ecore, es la entidad Application y engloba toda la plataforma. Dentro de cada una de las entidades, están los atributos.

Figura 27: Entidad Application

Directamente enlazadas de ella nos encontramos con las entidades *Template* y *Screen.* A partir de Template, es posible localizar todos los elementos que la forman. Este tipo de objeto se usa en la aplicación para establecer cómo será visualmente la pantalla y a qué pantallas podremos ir a partir de la entidad.

Una Template, que a su vez está relacionada con la entidad Screen, está compuesta por algunas entidades como son Head, Image o Footer

Éstas a su vez, están formadas por otras entidades que les van proporcionando atributos y así se van entrelazando los objetos.

Por otro lado, en la entidad Screen se representan los diferentes tipos de pantallas que podemos encontrar en la aplicación:

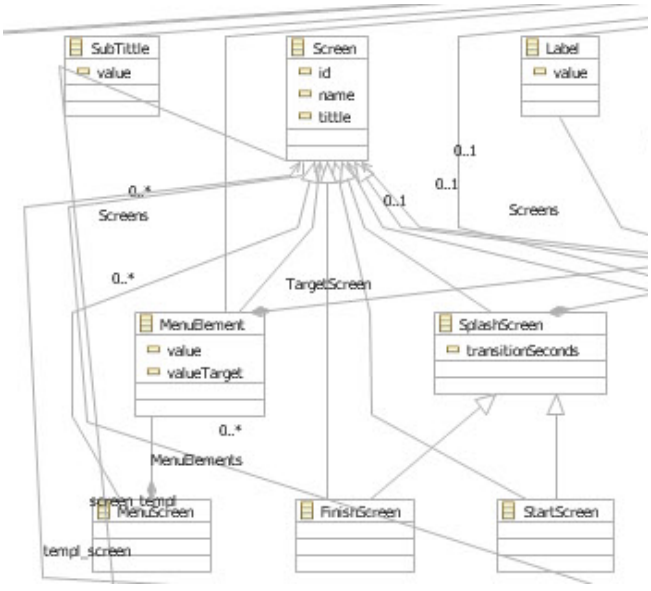


Figura 28: Entidad Screen

A partir de ella podemos encontrarnos con:

SplashScreen: Representa la pantalla inicial de la aplicación a generar.

FinishScreen: Representa la pantalla final de la aplicación a generar.

MenuScreen: Representa las pantalla con enlaces de menú

RegularScreen: Representa las pantallas de contenido de la aplicación

El resto de elementos del metamodelo, representan los diferentes objetos que formarán las pantalla. Éstos se relacionan con *Element*, que representa el elemento común. Obsérvese en la siguiente figura la cantidad de relaciones que posee

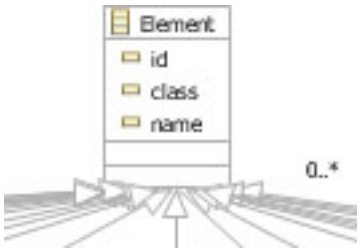


Figura 29: Entidad Element

4.3. Requisitos / Aspectos de la Interfaz del Usuario

Teniendo en cuenta las características de la plataforma a desarrollar, la interacción usuario – sistema debe de ser lo más amigable posible y más, si tenemos en cuenta que la plataforma también está basada en el desarrollo *End-User*.

Recordemos que este desarrollo que se define [29] como un conjunto de actividades o técnicas que permiten a las personas, que no son desarrolladores profesionales, crear o modificar software.

Surgió en respuesta a una iniciativa europea a partir del proyecto *EUD-Net* [2]: una Red de Excelencia Europea que tiene como objetivo hacer posible, de una forma rápida y barata, la co-evolución de sistemas y usuarios finales como uno solo, a partir de la adaptación explícita de las aplicaciones informáticas a las actividades de los propios usuarios. [29]

A grandes rasgos el principal objetivo que persigue el citado desarrollo es facilitar la autoría de software a cualquier tipo de usuario. Para ello se tienen en cuenta los siguientes hitos, que son al fin y al cabo los que se persiguen en MovilTurismo [29]:

2 http://www.eudnet.net/

Aumentar la participación del usuario en el proceso inicial de diseño: . Se trata de tener en cuenta la opinión del usuario en todo momento, haciendo un diseño mucho más centrado en el usuario y adaptado a sus propias necesidades.

Utilizar lenguajes visuales de modelado: Es primordial conseguir lenguajes fáciles de usar para el usuario, más intuitivos, así como lenguajes de dominio específico. Esto permite hacer frente a la diferencia de abstracción existente entre los profesionales del software y usuarios finales o expertos del dominio.

Llegar a un compromiso aceptable entre la expresividad y la facilidad de uso: Lo deseable para este caso es conseguir entornos de creación de software fáciles de utilizar, intuitivos, disminuyendo por el contrario la capacidad expresiva de los mismos.

Diseñar más software adaptativo: Teniendo al usuario final como objetivo en el diseño de software para autoría, sería conveniente, como ayuda al usuario, construir sistemas que se adapten a éste durante el uso. Es importante que el sistema lleve a cabo una adaptación no intrusiva, intentando no distraer el usuario de su tarea principal.

El resultado se deberá convertir en una interfaz intuitiva, sencilla y agradable que permitirá al usuario crear su aplicación con el menor esfuerzo posible.

Llegados a este punto, es necesario hablar de los mapas navegacionales:

Las propiedades navegacionales de una aplicación Web se describen [26] asociando un mapa navegacional para cada tipo de usuario.

Así, el mapa navegacional proporciona la vista global del sistema para cada uno de los tipos de usuario que disponga la aplicación y define la estructura del sitio Web.

Para poder explicar en qué consiste un mapa navegacional, nos basaremos en el siguiente ejemplo.

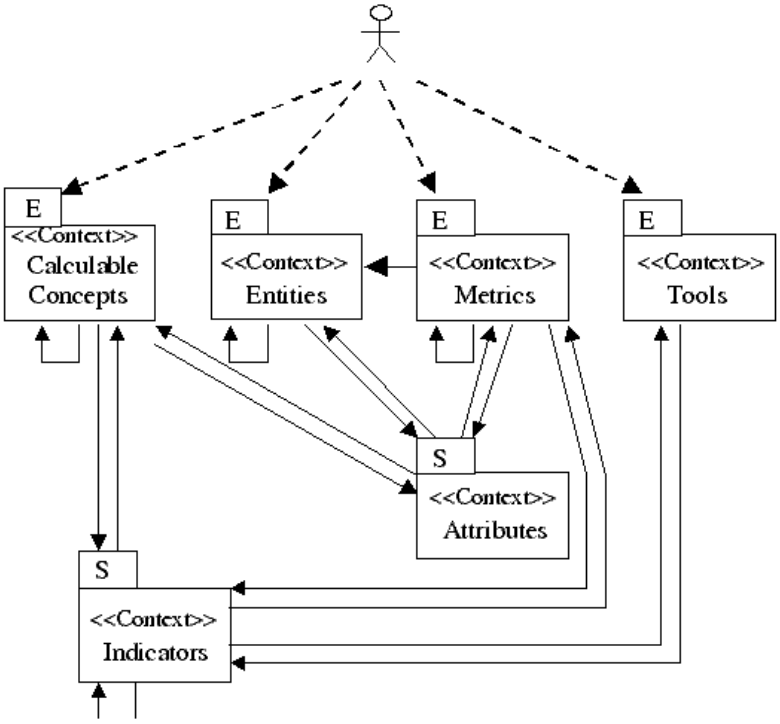


Figura 30: Ejemplo de mapa navegacional

Como podemos ver, en este mapa navegacional, podemos encontrar básicamente:

- **Nodos navegacionales**: Son los recuadros que aparecen, también conocidos como contextos.

  Un contexto es la unidad de interacción básica con el usuario. Internamente, se compone de clases navegacionales (operaciones que se pueden realizar) y relaciones navegacionales (cómo llegar a ellas).

  Además, un contexto puede ser de dos tipos:

  - *Exploración*: Están siempre accesibles por el usuario (E)
  - *Secuencia*: Sólo accesibles a través de caminos de navegación preestablecidos (S). Estos caminos, son enlaces a través de los cuales se puede navegar de una página a otra, o a través de los cuales, las páginas se envían información.

- **Enlaces navegacionales**: Las líneas que unen los contextos y el usuario con los mismos. También pueden ser de dos tipos:

  - *Exploración*: son las flechas discontinuas y están definidas por contextos de exploración.
  - *Secuencia*: son las flechas discontinuas. Definidas por las relaciones navegacionales del contexto.

Vemos a continuación los mapas navegaciones de MovilTurismo en función del tipo de usuario:

Usuario anónimo

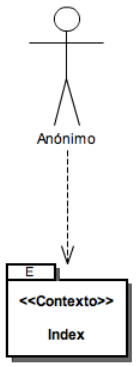


Figura 31: Mapa Navegacional usuario anónimo

Usuario Registrado

Debido al gran tamaño del mapa navegacional, lo hemos separado en dos partes. Una parte representa los contextos accesibles desde el perfil y el otro, el resto de la aplicación. El punto de unión es el contexto "Generación"

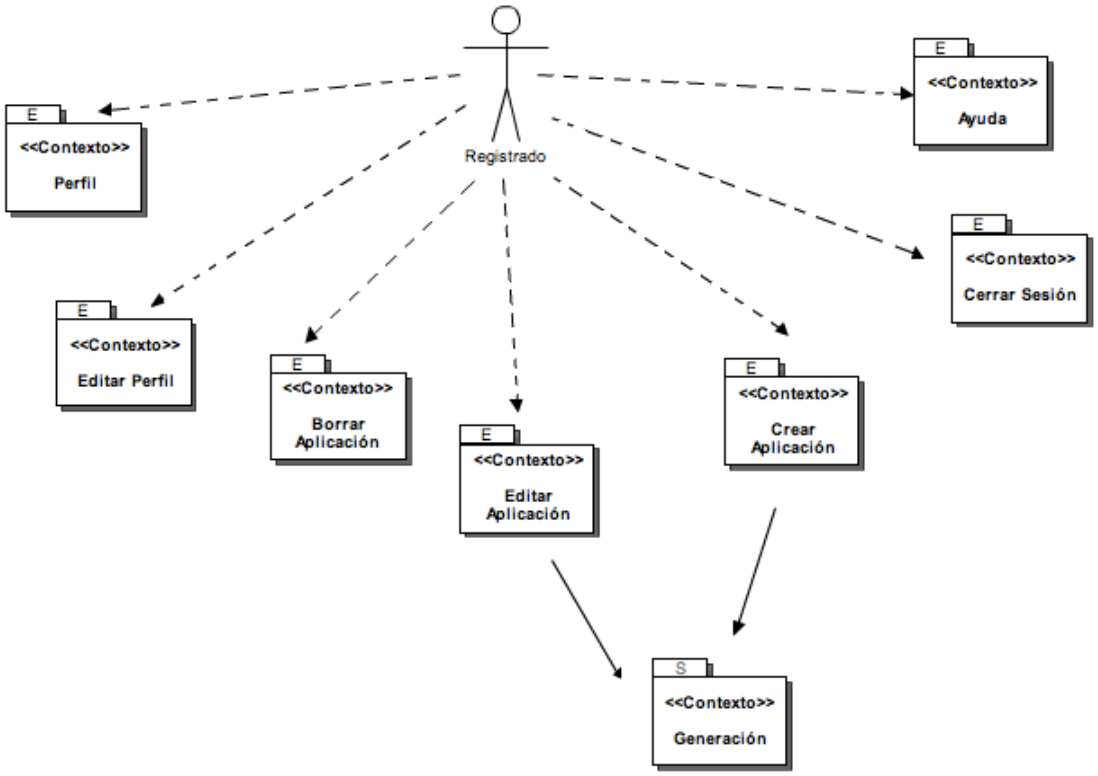


Figura 32: Mapa Navegacional usuario Registrado (I)

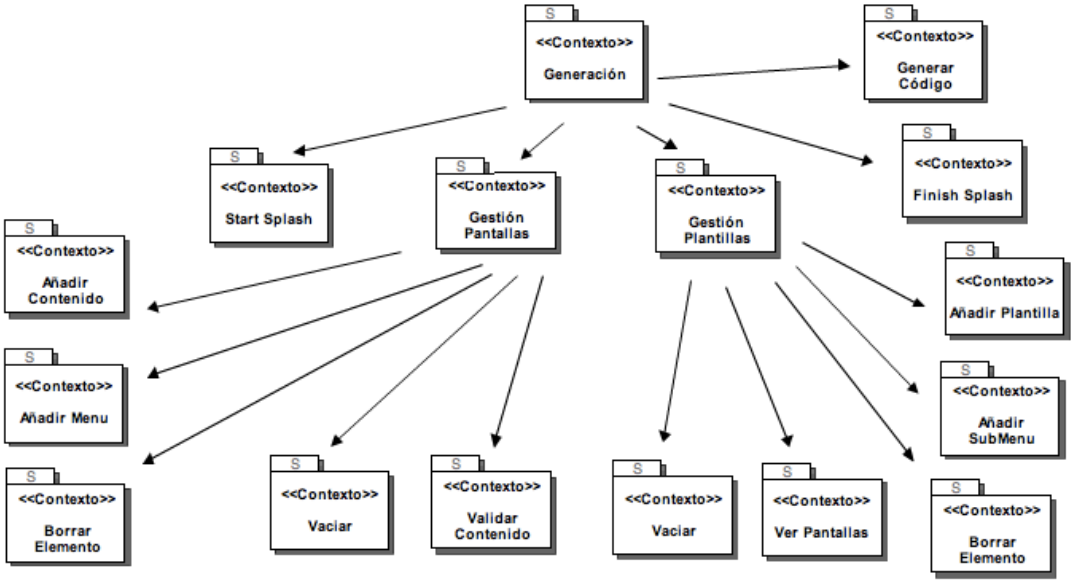


Figura 33: Mapa Navegacional usuario registrado (II)

5. Implementación de MovilTurismo

El desarrollo de MovilTurismo ha sido realmente complejo, debido a la cantidad de tecnologías diferentes a utilizar en el proyecto

Realizar este proyecto ha supuesto solventar varios problemas que han ido surgiendo a lo largo del desarrollo. En primer lugar, debía solventarse cómo una aplicación Web iba a ser capaz de generar código funcional usando MDA.

Pero vayamos por partes: para gestar el proyecto, fue necesaria una versión previa con GMF que sirvió como base para la actual plataforma. En otras palabras, existen principalmente 2 versiones distintas que se han complementado para llevar a cabo el desarrollo.

A lo largo de este punto, vamos ir haciendo distinción entre ambas versiones, por lo que en el punto 5.1, hablaremos sobre la tecnología empleada en ellas. En el punto 5.2, veremos las interfaces de ambas versiones mencionadas anteriormente. El punto 5.3 mostrará como se ha creado el generador, el 5.4 mostrará cómo se modifican el documento XMI del modelo con JSF y por último el punto 5.5 recogerá los resultados y el 5.6 las conclusiones.

5.1. Desarrollo MDD

Hemos comentado que existen 2 versiones que han ido apoyándose conjuntamente para crear la plataforma actual.

La primera versión, se gestó con GMF y la segunda con JSF. El resultado con la primera versión fue el esperado, sin embargo no resultó ser nada usable para el usuario final. Vamos a ver cada una de ellas.

5.1.1. Primera Versión: MovilTurismo / GMF

Actualmente, la base funcional de MovilTurismo, se apoya firmemente en un proyecto previo realizado en la asignatura *Patrones Software y Generación de código* del presente Máster que recibió el nombre "*Portaventura Guide*".

A groso modo, en dicho proyecto se consiguió generar una guía turística totalmente funcional para la plataforma Blackberry, usando la generación de código por modelos.

En ese caso en concreto, se eligió el domino de las "guías turísticas" para realizar el desarrollo. Dada la naturaleza de la generación por modelos, podríamos generar cualquier tipo de aplicación en este dominio, así como modificar una existente.

La generación de la aplicación, podría resumirse en *"generar un editor gráfico y transformar con plantillas a partir de éste"*

A continuación, mostramos el proceso tal y como se siguió:

La versión partió de un metamodelo (.ecore) ya creado (prácticamente idéntico al del Anexo 2). El metamodelo permite reflejar a simple vista, los componentes que tendrá la aplicación que va a generarse y visualmente es similar a un diagrama UML.

Así pues, una vez con el metamodelo debidamente definido, el siguiente paso consistió en pasar ese metamodelo a GMF[3] realizando varias transformaciones sobre él. Para realizar todo este proceso, así como el venidero, se usó la siguiente herramienta:

5.1.1.1. MOSKitt

Modeling Software KIT (MOSKitt) [30] es una herramienta CASE LIBRE, basada en Eclipse que está siendo desarrollada por la Conselleria de Infraestructuras, Territorio y Medio Ambiente para dar soporte a la metodología gvMétrica [4], la cual usa técnicas basadas en el lenguaje de modelado UML.

Su arquitectura de plugins la convierte no sólo en una Herramienta CASE sino en toda una Plataforma de Modelado en Software Libre para la construcción de este tipo de herramientas.

[3] GMF es un Framework de la plataforma Eclipse que proporciona un componente de generación y una infraestructura para el desarrollo de editores gráficos basados en EMF y GEF

[4] http://www.gvpontis.gva.es/fileadmin/conselleria/images/Documentacion/migracionSwAbierto/gvMETRICA/introduccion/gvMetrica_Introduccion.htm

Figura 34: Logo MOSkitt

El *Centro de Investigación en Métodos de Desarrollo de Software* [5](*ProS*) de la Universidad Politécnica de Valencia, ha desarrollado MOSKitt Feature Modeler (MFM), una herramienta gratuita y open-source basada en Eclipse para dar soporte a los modelos de características.

MFM puede utilizarse de forma independiente como un plug-in de Eclipse o integrada en el entorno MOSKitt.

5.1.1.2. GMF

El Framework de modelado gráfico (Graphical Modeling Framework) de Eclipse, proporciona un componente "generador" y una infraestructura sólida para el desarrollo de editores gráficos basados en EMF y GEF [6].

La construcción del editor gráfico consta de tres pasos básicos:

- Definición Gráfica
- Definición del "Tooling"
- Definición del "Mapping"

Veámoslo de manera gráfica[7]:

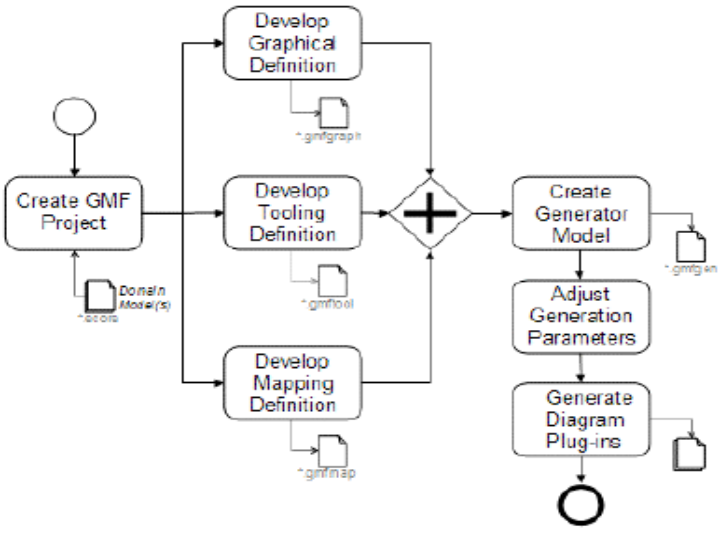


Figura 35: Esquema

[5] http://www.pros.upv.es/index.php

[6] Más información en: http://www.eclipse.org/articles/Article-GEF-EMF/gef-emf.html

[7] Más información en: http://www.eclipse.org/modeling/gmp/

Como se observa en la Figura 35 la creación de un nuevo proyecto GMF, implica la creación de los 3 puntos anteriormente mencionados. Una vez se han creado, todos se unen para crear el "*Generator Model*" y a partir de ahí, mediante una serie de plantillas, poder generar el código final. Pero primero, vayamos parte por parte y veamos el proceso paso a paso.

ecore (Domain Model)

El .ecore es el archivo principal del proyecto, ya que contiene el metamodelo que definirá la aplicación. De hecho, este archivo es necesario para crear el nuevo proyecto GMF .

Como se ha mencionado anteriormente, pare realizar el proyecto se partió de un archivo ya creado y a partir de él, se generaron el resto de archivos necesarios para obtener el código final.

Mostramos a continuación un fragmento de dicho archivo. Cada una de las ramas que se pueden apreciar en la Figura 36 representa una clase del diagrama UML. Desplegando cada una de las ramas es posible acceder a los atributos de cada clase.

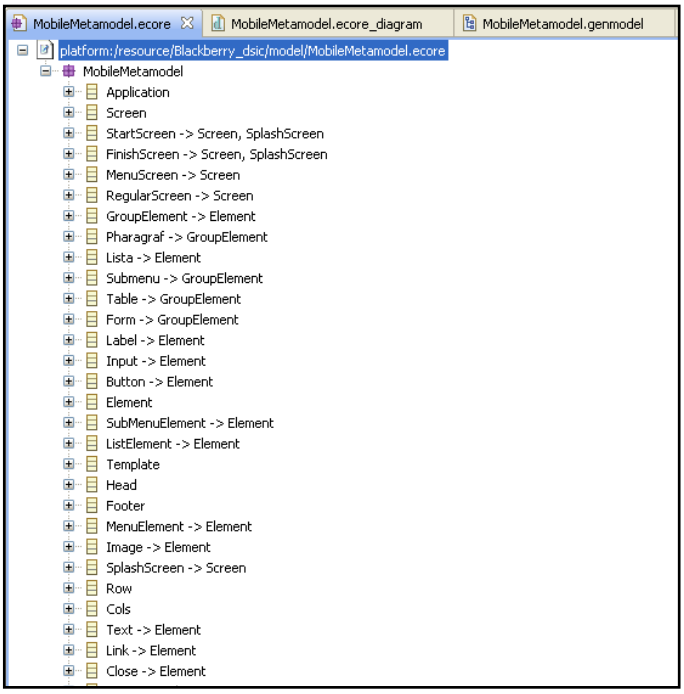


Figura 36: Aspecto del fichero ecore

gmftool (Tooling)

En este fichero se define la creación de la paleta de dibujo, que tendrá el editor gráfico. Cada una de las "*Creation tool*" definidas en este fichero se traducirán como herramientas en la paleta del editor.

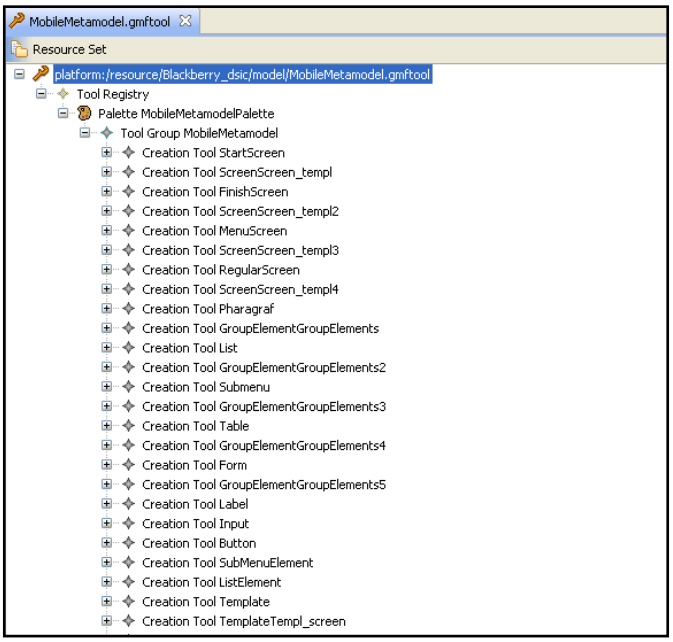


Figura 37: Aspecto del fichero gmftool

gmfgraph (Definición Gráfica)

Es el encargado de que se puedan dibujar los objetos del modelo dentro del área de dibujo.

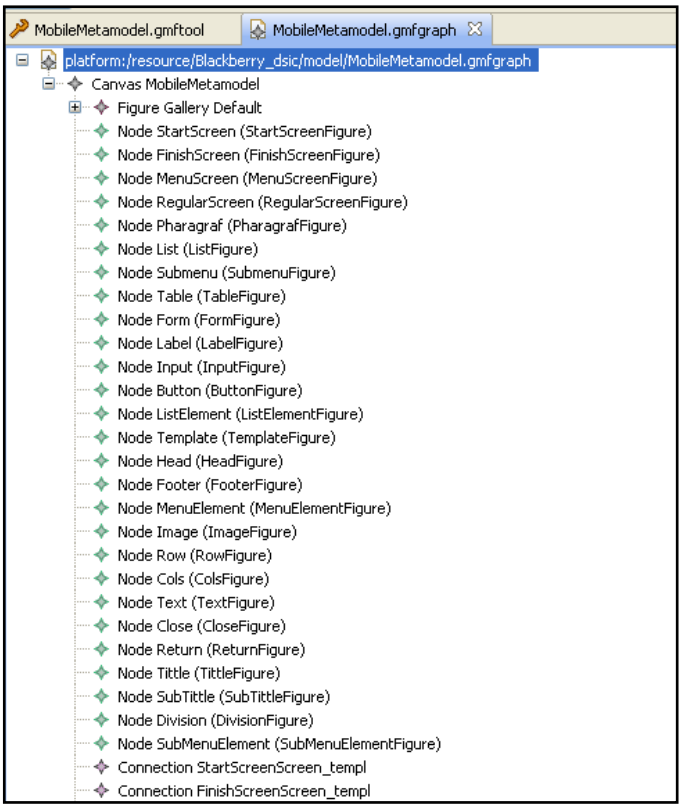


Figura 38: Aspecto del fichero gmfgraph

Como vemos en la imagen, la definición gráfica está formada por diversos elementos. En sus estructura de árbol podemos ver un elemento “Figure Gallery”, que contiene la figura en sí y como se verá en el editor.

El resto de elementos del árbol son los nodos del diagrama UML y los *compartments*, necesarios para poder dibujar nodos dentro de nodos.

Si desplegamos el nodo "*Figure Gallery*", vemos que contiene las propias figuras que se dibujarán en el editor, y dentro de éstas, sus características (si será un rectángulo, el color de fondo, etc.…)

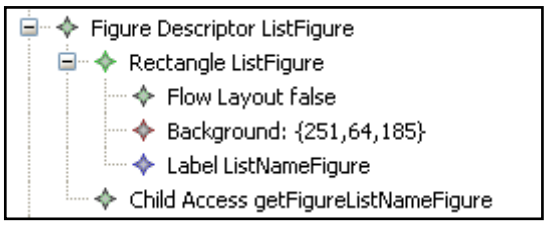


Figura 39: Ejemplo de Características

gmfmap (Mapping)

Define las correspondencias entre el modelo de dominio y el modelo gráfico (.gmfgraph).

Crea un nuevo modelo con los contenidos de la definición gráfica y la definición de la paleta. La estructura utilizada es la misma que en los otros ficheros, una estructura arbórea fácil de modificar e intuitiva de visualizar.

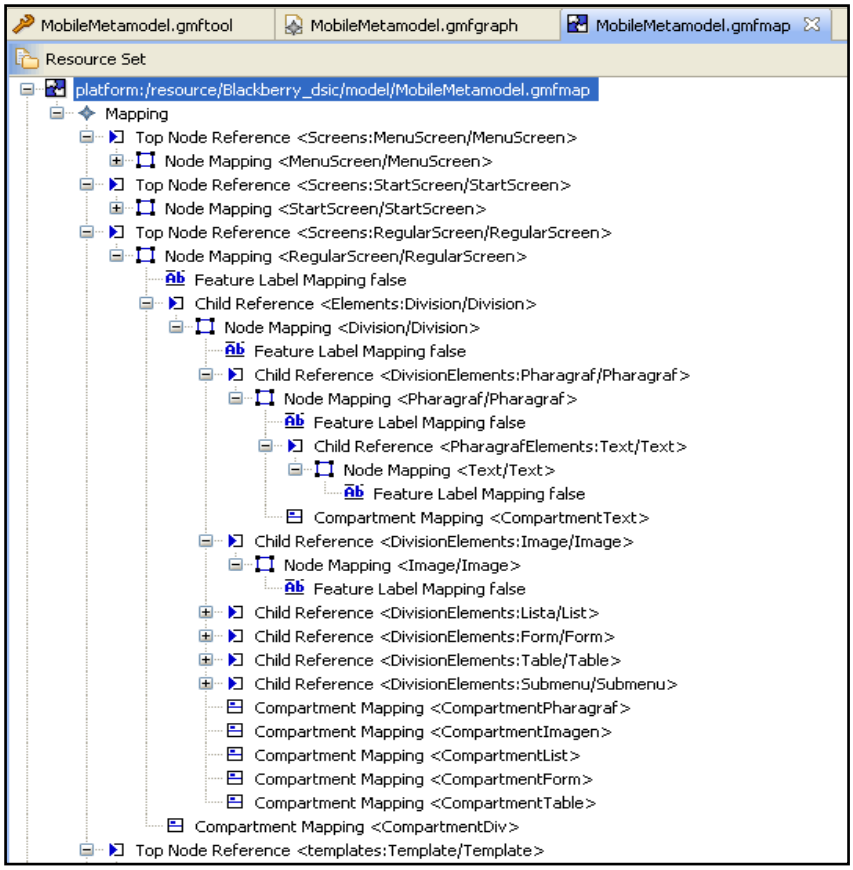


Figura 40: Aspecto del fichero gmfmap

Así pues, el modelo citado anteriormente, contiene en su estructura de árbol todas las definiciones que integra, de tal manera que por ejemplo un cambio en la paleta, se verá

reflejada automáticamente sin necesidad de volver a generar nada.

Aquí, se especifica básicamente la manera en que deben dibujarse los elementos y dentro de cuáles.

Por ejemplo, para definir que dentro de un elemento "*division*" pueden incluirse una serie n de elementos (por ejemplo párrafo o imagen), la estructura que deberíamos generar sería de la forma siguiente:

```
Top Node Reference (regularScreen)
   Node Mapping (regularScreen)
         Label
         Child Reference (division)
               Node Mapping(division)
                     Label
                     Child Reference(elemento 1)
                           NodeMapping (elemento 1)
                           Label
                     Child Reference(elemento 2)
                           NodeMapping (elemento 2)
                           Label
                     .
                     .
                     .
                     Child Reference(elemento n)
                           Node Mapping (elemento n)
                           Label
                     Compartment CompartmentElemento1
                     Compartment CompartmentElemento2
                     .
                     .
                     Compartment CompartmentElementon
```

Figura 41: Ejemplo gmfmap

En el esquema podemos ver que cada uno de los elementos tiene un "compartment", este elemento definido en el archivo .gmfgraph, es el "Contenedor" que engloba el elemento que vamos a dibujar.

El compartment por ejemplo nos permite dibujar un elemento imagen dentro de un elemento pantalla, o un elemento texto dentro de un elemento párrafo (cada uno de esos elementos tiene su compartment definido)

gmfgen (Generator Model)

Es el primer fichero que no se genera desde el .ecore, se genera desde el .gmfmap y sirve para crear el generador de modelos, es decir, el editor gráfico.

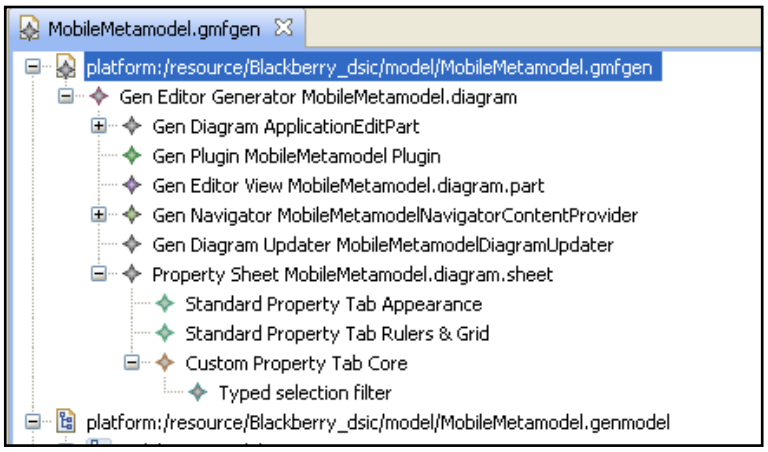


Figura 42: Aspecto del fichero gmfgen

Editor gráfico

Creado este último fichero, para lanzar a ejecución la aplicación, debe ejecutarse como "Eclipse Application".

Con la segunda instancia de MOSKitt en marcha, creamos un proyecto simple vacío, y en él, creamos la instancia del modelo que hemos ido preparando. Podemos seleccionar la instancia desde (y sobre el proyecto) New/Others/Examples

Seguidamente se mostrará el editor en pantalla. A la derecha veremos la paleta de dibujo con las herramientas que hemos definido y en el centro el área para dibujar. De esta manera, podremos ir creando nuestra aplicación de manera gráfica.

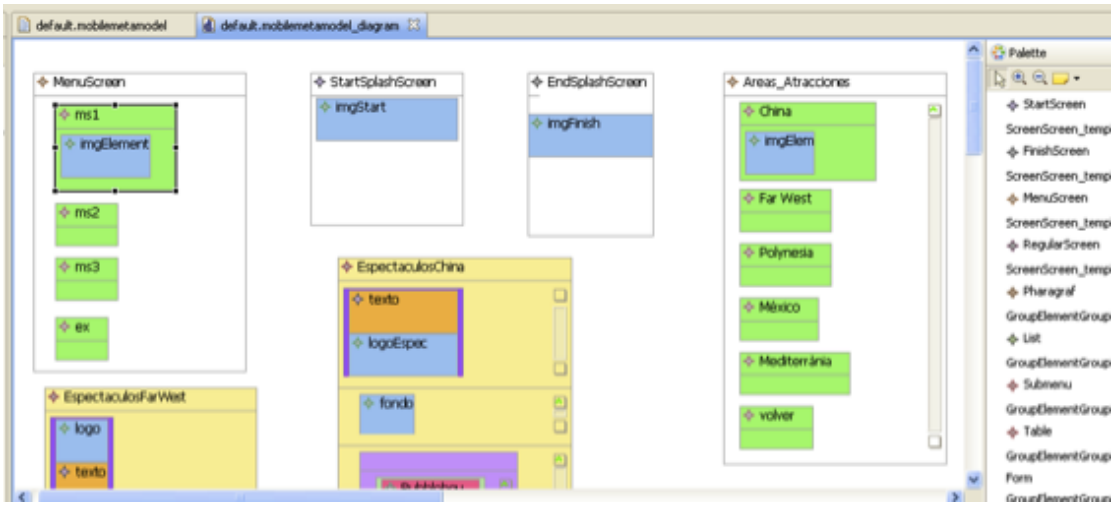


Figura 43: Aspecto del editor gráfico

Esto genera un fichero *default.mobilemetamodel* en formato XMI en el que se recoge toda la aplicación.

5.1.2. Segunda Versión: MovilTurismo / JSF

La creación de esta segunda versión resulto ser más compleja que la primera, si bien los resultados obtenidos fueron superiores.

La solución se basó en intentar emular el funcionamiento de la suite MOSKitt que se usa en la primera versión a través de una aplicación Web así como usar Xpand2 para las transformaciones. Esta emulación resultó bastante compleja ya que estamos hablando de sustituir toda la generación de una suite completa en una única página Web.

Realizar este proyecto ha supuesto solventar varios problemas que han ido surgiendo a lo largo del desarrollo. En primer lugar, debía solventarse cómo una aplicación Web iba a ser capaz de generar código funcional usando MDA.

Para poder realizar la generación, se importaron al proyecto las mismas librerías que usa MOSKitt para generar código, sin embargo, tocó enfrentar el siguiente reto, ¿cómo poner en marcha el mecanismo de transformación?

Para ello se recurrió directamente a la fuente [24] y se encontró un ejemplo de código que proporcionaban para poder arrancar el mecanismo. Afortunadamente el código proporcionado era Java, con lo que fue posible adecuarlo a las necesidades del proyecto.

```
String wfFile = "somePath\\workflow.oaw";
Map properties = new HashMap();
Map slotContents = new HashMap();
new WorkflowRunner().run(wfFile ,
    new NullProgressMonitor(), properties, slotContents);
```

Fueron necesarias muchas pruebas para verificar que era posible aprovechar esta tecnología desde una aplicación Web. Estas pruebas consistían en crear un botón con JSF desde MovilTurismo, que llamara internamente al fichero *default.mobilemetamodel*, que recordemos, era el fichero del editor gráfico que generaba MOSKitt al realizar la generación de código. Para consultar este fichero con más detalle, leer el punto 5.3 del presente documento.

Este fichero, se tomó de creación de la versión anterior y se ubicó en la carpeta TMP del equipo. Adicionalmente para la creación, es necesario que el fichero de metamodelo se encuentre en la carpeta del workspace del proyecto. El fichero de plantillas lo coge del propio proyecto (está incluido en él).

Dadas las nuevas características de la segunda versión, ha sido necesario crear una plantilla adicional a la de la primera versión para realizar el parseado para las aplicaciones que deban ejecutarse en HTML5. Ambas plantillas pueden consultarse en el Anexo 4 del presente documento.

De esta manera, se volvieron a generar las clases Java de *Portaventura Guide* en la carpeta TMP del ordenador con lo que se dieron por buenas las pruebas y se consideró apto para emular a MOSKitt.

Mostramos un ejemplo ilustrativo:

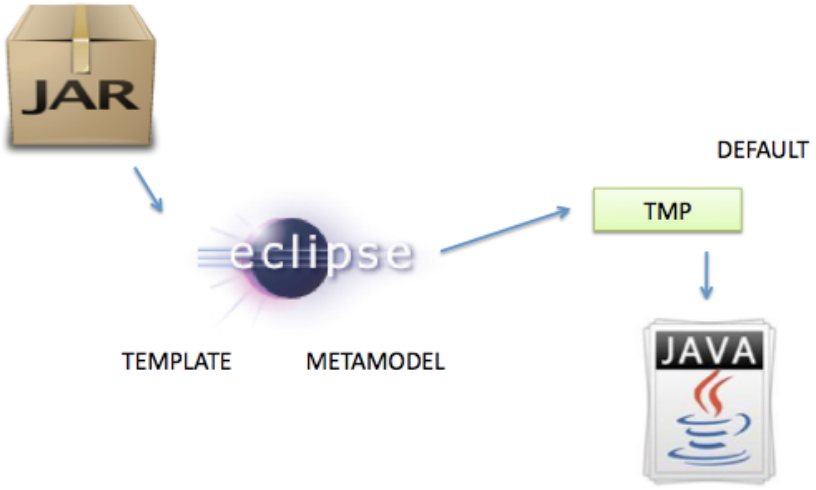


Figura 44: Prueba generación 1

En este punto, contamos con el mecanismo de generación a punto y se había comprobado que era posible generar código a partir de una aplicación Web. Esto nos llevaba al siguiente punto ¿cómo generar el fichero *default.mobilemetamodel*?

La idea fue crear un fichero cuyo contenido guardara la misma estructura que el fichero default (en formato XMI). La aplicación almacenaría la información de la configuración de la aplicación en curso que estuviese creando el usuario en buffers y los volcaría al fichero respetando la estructura necesaria. Este volcado es muy complejo y puede verse con más detalle en el punto 5.4

Por supuesto toda la información de la aplicación en curso, debería almacenarse en algún sitio, por lo que se eligió MySQL como motor de base de datos y se diseñó la base de datos necesaria para almacenar toda la información necesaria de las pantallas. Puede verse esta información con más detalle en el punto 4.5.2 del presente documento.

Una vez con la parte principal más o menos perfilada, el siguiente paso fue centrarse en la funcionalidad de la propia aplicación, así como de su apariencia.

La librería *PrimeFaces* ofrece multitud de elementos y componentes para situar en nuestro sitio Web. Es una librería en constante crecimiento y en desarrollo y es lógico que presente ciertos errores (en la actualidad trabajan para solventarlos).

En un primer momento y partiendo de un ejemplo dado, se intentó diseñar la interfaz gráfica con PrimeFaces 1.2. Este hecho obligaba a desarrollar con la versión 1.2 de JSF y con JSP.

El desarrollo estuvo enfocado a estas versiones durante mucho tiempo, hasta que la gestión de pantallas propia de MovilTurismo nos obligó a cambiar de versión: el componente de PrimeFaces que debía usar y que visualmente es una estructura de árbol, presentaba multitud

de errores en una versión más temprana y no fue posible adecuar la aplicación a la funcionalidad que funcionaba correctamente en ese momento.

Por ello, nos vimos obligados a migrar la aplicación a la versión más estable de PrimeFaces, en ese momento la versión 2. Este cambio no resultó para nada trivial, puesto que implicaba también el cambio de la versión de JSF a la 2.0

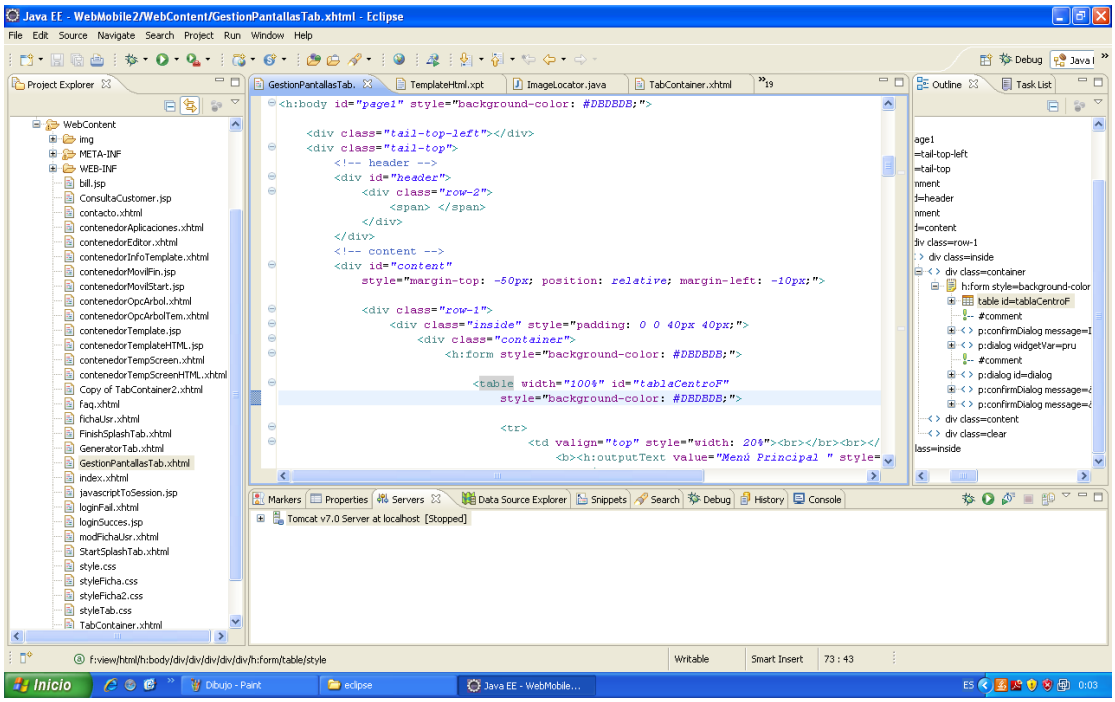


Figura 45: IDE Eclipse

Además, para esta versión de JSF no existe compatibilidad con JSP con lo que obligó una vez más, a cambiar la mayoría de los archivos JSP por archivos XHTML con el retraso que ello supuso.

Tras todo esto, veamos la estructura del proyecto:

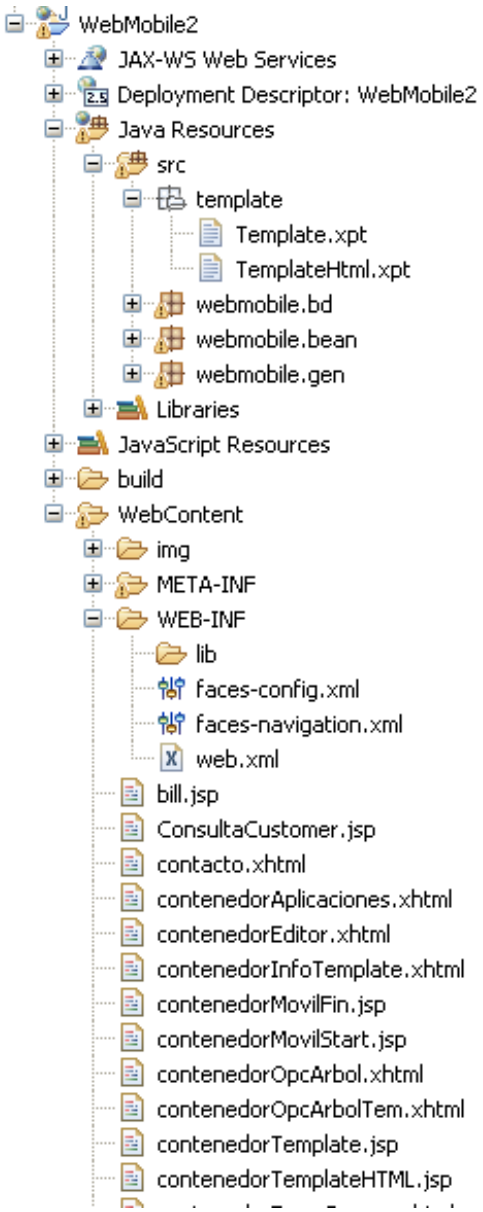


Figura 46: Estructura del proyecto MovilTurismo

La carpeta *src* es la que contiene la lógica de la aplicación. En ella distinguimos diferentes paquetes:

- *template*: Contiene las plantillas de conversión.
- *webmobile.bd*: Incluye las clases Java encargadas de interaccionar con la base de datos.
- *webmobile.bean*: Incluye los Beans de la aplicación
- *webmobile.gen*: Incluye el resto de clases Java que no han sido clasificadas en los anteriores paquetes.

Por otro lado, la carpeta *WebContent*, incluye la representación visual de la aplicación y las configuraciones propias de JSF. Destacamos la carpeta img, que contiene todas las imágenes que se muestran en MovilTurismo, la carpeta *WEB-INF* con los archivos necesarios de *JSF* y el resto de archivos web *XHTML* y *JSP*.

5.2. Interfaz gráfica

Realmente, para la primera versión con GMF, no existe una interfaz gráfica de cara al usuario final mas allá de la que pueda proporcionarnos MOSKitt o Eclipse para establecer las propiedades que va a tener la aplicación a generar.

El usuario dispone de estas suites para generar su aplicación, pero dada la complejidad y la ausencia de una interfaz amigable que les facilite un poco el trabajo, no resulta nada usable.

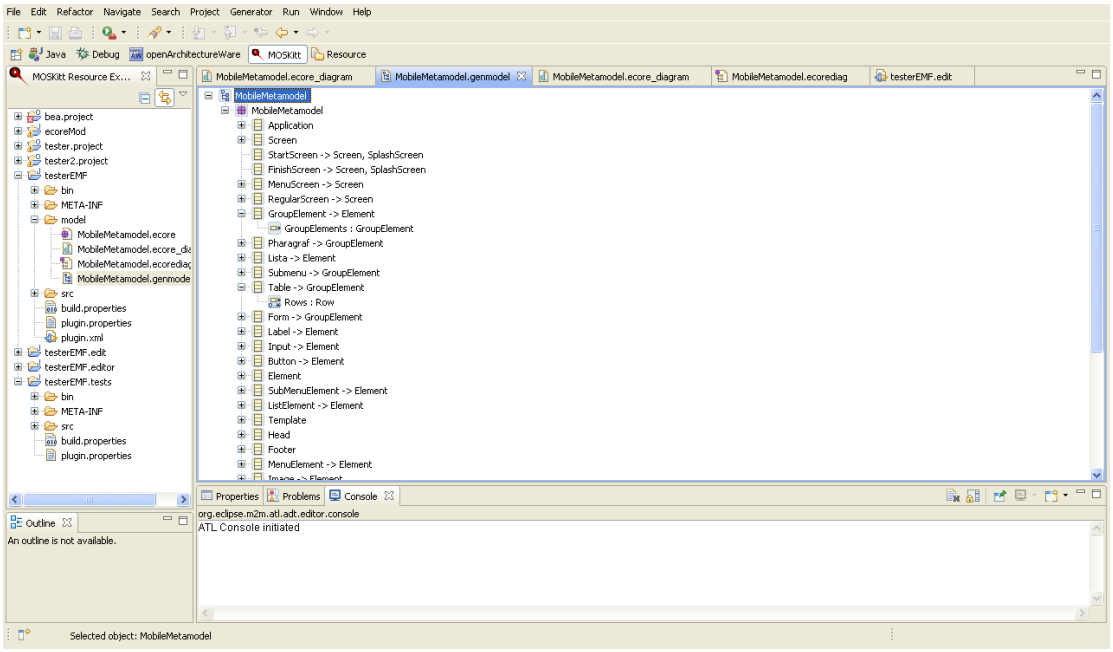


Figura 47: Interfaz versión GMF

Sin embargo, el fuerte de la segunda versión con JSF es justamente eso: la interfaz gráfica.

MovilTurismo dispone de una interfaz gráfica agradable a la vista, sencilla, amigable y usable, que proporciona ayuda en todo momento al usuario y que le permite controlar en todo momento qué está creando y cómo se va a ver.

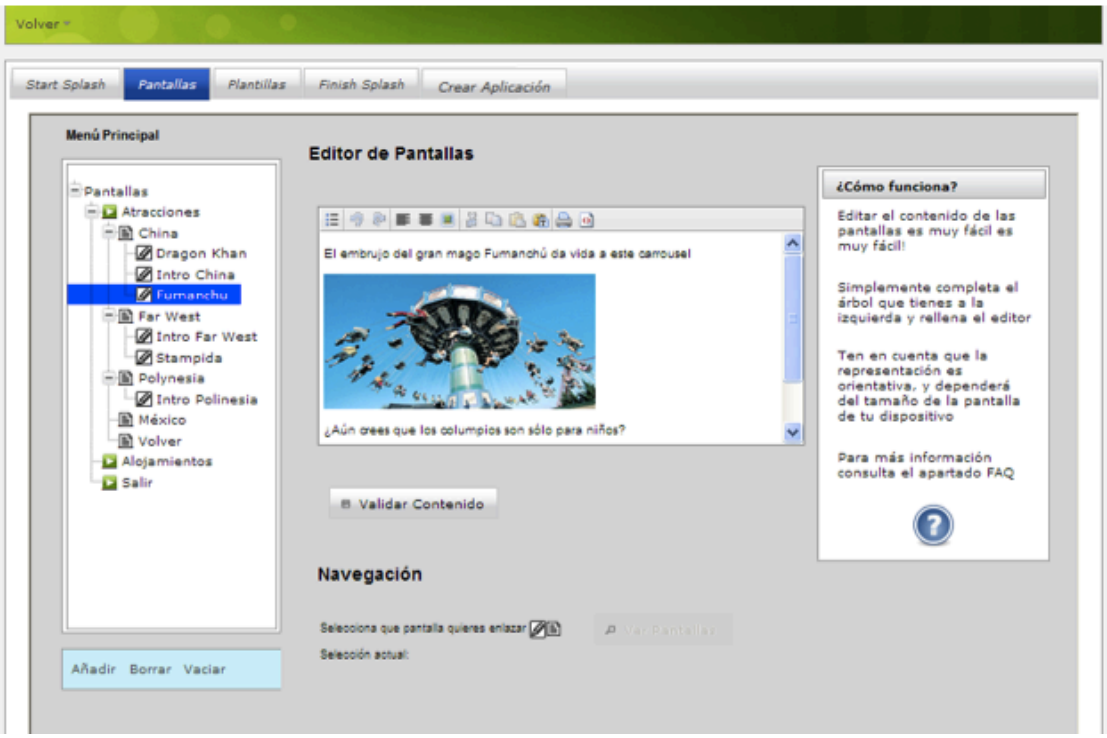


Figura 48: Interfaz Versión JSF

Como tónica general, se le solicita al usuario introducir los datos y crear las pantallas en base a un componente árbol. Este componente les permite visualizar en todo momento, la estructura de la aplicación que están creando. Este mecanismo también lo usa **TEKORA**.

5.3. Implementación de los generadores

En el siguiente punto, vamos a ver cómo él es funcionamiento del mecanismo de conversión en la versión de JSF. Para ello hemos dividido el apartado en tres partes: *Generando código con Xpand2*, *El fichero default.mobilemetamodel* y *Conversión.*

Pero antes, vamos a ilustrarlo en el siguiente esquema:

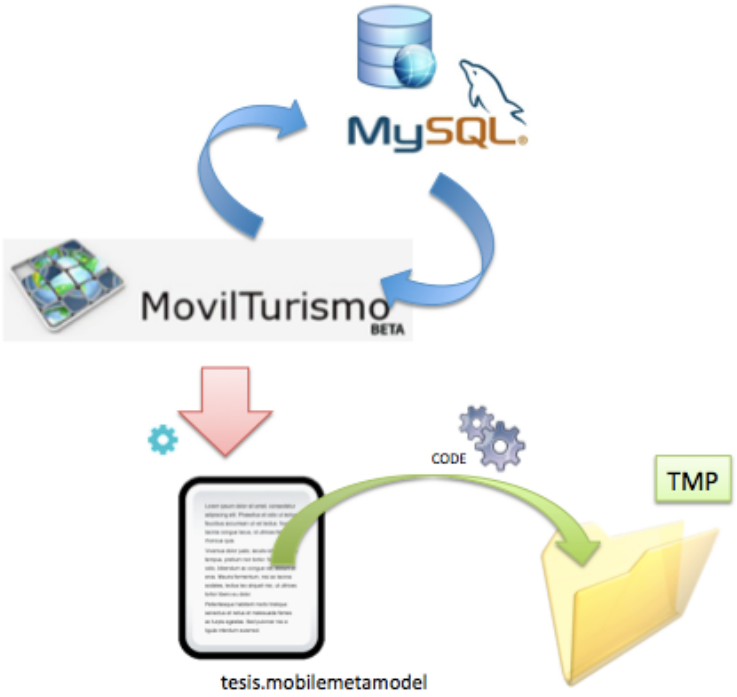


Figura 49: Mecanismo de generación de MovilTurismo

Una vez que el usuario ha acabado de crear la aplicación y pone la marcha el mecanismo, MovilTurismo realiza accesos a base de datos para recuperar la información de la aplicación en curso.

La información recibida es procesada y "parseada" de tal manera que cumpla con el formato del fichero default.mobilemetamodel que tenemos de ejemplo. El nuevo fichero recibe el nombre de *tesis.mobilemetamodel*

Con el nuevo fichero gráfico listo y la plantilla oaw correspondiente, se genera el código y se guarda en la carpeta TMP del ordenador.

5.3.1. Generando código con Xpand2

5.3.1.1. OAW

OAW es un Framework modular (de código abierto) para el parseado de Modelo que dispone de una familia de lenguajes para comprobar y transformar modelos, así como generar código a partir de ellos.

Para ello, dicho Framework proporciona diferentes editores que están basados en Eclipse.

Principalmente está orientado a dar soporte a modelos definidos en EMF, aunque puede soportar otros modelos, como UML o XML.

El Framework OAW se compone de los siguientes elementos:

- Un *workflow engine* (motor de flujos de trabajo), que permite indicar todos los trabajos que se deben realizar y en qué orden deben hacerse
- Un lenguaje de restricciones.
- Un lenguaje para las transformaciones de modelo a modelo: *Xtend*
- Un lenguaje para las transformaciones de modelo a texto: ***XPand2***

Xpand2, está contenido dentro del Framework modular de OpenArquitectureWare (OAW) y es un lenguaje de plantillas para controlar la generación de código.

Este paquete está integrado en Eclipse, es de código abierto y proporciona soporte para EMF y UML2[8].

Su funcionamiento es el siguiente:

Una plantilla recibirá como entrada uno o varios modelos (suministrados desde el fichero del workflow). Las transformaciones a dichos modelos se realizarán en base a la aplicación de unas reglas definidas en la plantilla:

- Cada regla llevará asociado un contexto (Elemento del metamodelo al que la regla es aplicable)
- Desde el fichero del workflow se indicará la primera regla a invocarse y sobre qué elemento del modelo suministrado.
- La salida será texto por pantalla o ficheros de texto.

Este es el esquema de una plantilla:

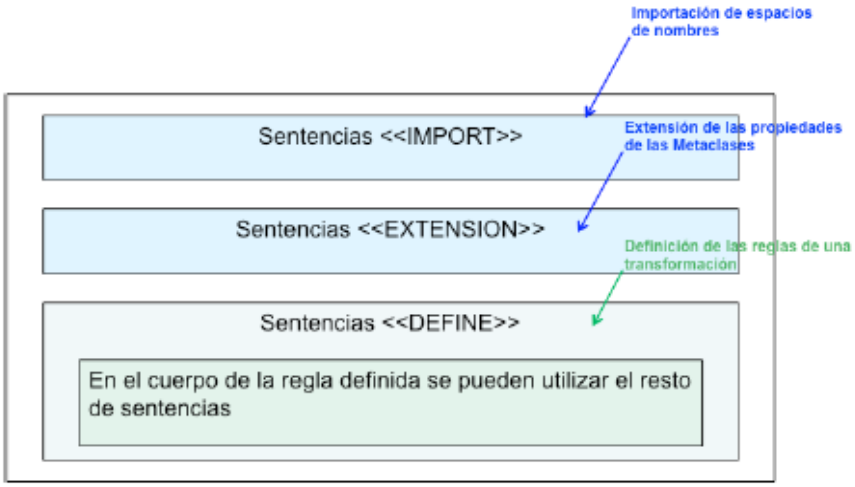


Figura 50: Esquema de una plantilla

Y este es el esquema del proceso completo:

[8] Más información en: http://www.eclipse.org/modeling/mdt/?project=uml2

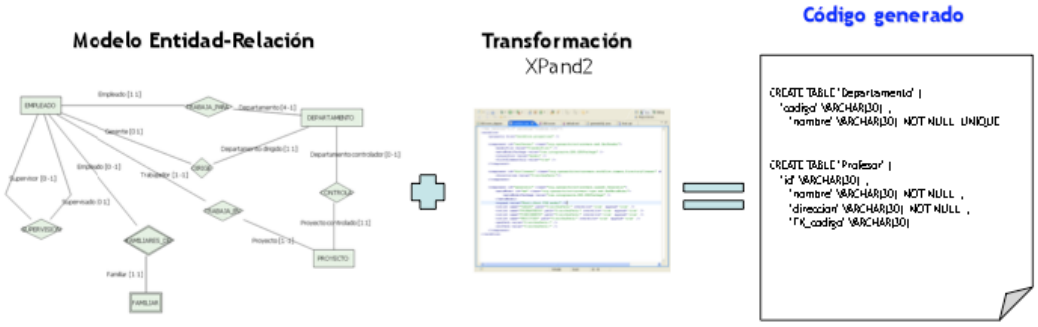


Figura 51: Esquema del proceso

Para empezar a transformar, primero fue necesario bajarse los plugins necesarios y seguidamente crear un nuevo proyecto OAW , con la siguiente estructura:

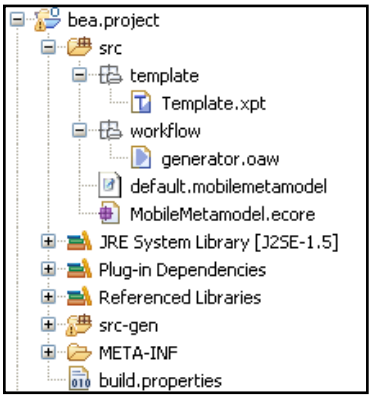


Figura 52: Proyecto OAW

Destacamos de esta estructura el archivo generator.oaw, que es el que define el flujo de trabajo principal para realizar la transformación. Aquí se define tanto la plantilla raíz (la primera en ejecutarse) así como otras opciones de configuración.

El contenido del fichero *generator* era el siguiente:

```
<workflow>

    <bean class="org.eclipse.mwe.emf.StandaloneSetup">
        <platformUri value=".."/>
        <registerEcoreFile value="src/MobileMetamodel.ecore" />
    </bean>


    <component id="read" class="org.eclipse.mwe.emf.Reader">
        <modelSlot value="model" />
        <uri value="src/default.mobilemetamodel" />
    </component>


     <component class="org.openarchitectureware.xpand2.Generator">
        <!--<metaModel id="mm" class="org.eclipse.m2t.type.emf.EmfRegistryMetaModel">-->
        <metaModel id="mm" class="oaw.type.emf.EmfMetaModel">
        <!--<metaModelPackage value="org.eclipse.emf.ecore.EcorePackage"/>-->
        <metaModelFile value="src/MobileMetamodel.ecore" />
        </metaModel>
        <expand
            value="template::Template::principal FOR model" />
        <outlet path="src-gen" />
    </component>
</workflow>
```

Figura 53: Contenido generator.oaw

También destacamos el fichero *Template.xpt* en el que se encuentran definidas todas las reglas para generar el código Blackberry.

Ésta plantilla recibe con entrada un modelo suministrado desde el fichero del workflow (y que se genera con el editor gráfico) y las sentencias principales que la definen son (entre otras):

- IMPORT
- DEFINE
- EXPAND

```
«IMPORT MobileMetamodel»

«DEFINE principal FOR Application»
    «FILE "Application.java"»

    import net.rim.device.api.ui.UiApplication;

    class Application extends UiApplication {

    Application() {
       pushScreen(new StartSplashScreen(this,new MenuScreen(this)));
    }

     public static void main(String[] args){
       Application myApp = new Application();
       myApp.enterEventDispatcher();
    }
}



    «ENDFILE»


    «EXPAND claseJava FOREACH Screens»
    «EXPAND claseTemp FOREACH templates»

«ENDDEFINE»

«DEFINE claseJava FOR Screen»
    «FILE name+".java"»
```

Figura 54: Contenido Template

La idea es la siguiente:

El fichero generator.oaw carga el modelo que hemos dibujado/generado en el editor gráfico. Una vez cargado, a partir de la plantilla principal especificada, oaw recorre el modelo dado y por cada uno de los elementos que contiene, mira si en la plantilla existe alguna definición para ese elemento.

En caso afirmativo, ejecuta el contenido de la sentencia DEFINE asociada. Este contenido puede ser por ejemplo, crear un nuevo archivo java para ese elemento y en función de a qué tipo pertenezca crear unas sentencias de código u otras

El proceso continúa ejecutándose y se da por finalizado cuando ha recorrido todos los elementos del modelo dado.

Si la transformación ha sido correcta, aparecerá en la carpeta src-gen del proyecto las clases generadas, listas para ejecutar.

5.3.2. El fichero default.mobilemetamodel

Hemos mencionado anteriormente el fichero *default.mobilemetamodel*, y lo hemos definido como el fichero gráfico que genera MOSkitt y a partir del cual se genera al código de la aplicación.

Dada su vital importancia para la generación de código, vamos a verlo más en profundidad:

¿Cómo se generaba este fichero? Si recordamos el punto 3.2.2.2 del presente documento, al ejecutar el proyecto GMF con MOSKitt nos ofrecía la posibilidad de "dibujar nuestra aplicación". La representación visual de es como un "paint" donde los elementos disponibles para dibujar son los que se han ido definiendo en el proyecto GMF. (Ver figura 43)

En lo que respecta a la parte no visual para el usuario, podríamos decir que el fichero gráfico es un gran XML donde van definidos los elementos y las características que son parte de ellos. Estas características nos las ofrece el fichero de metamodelo.

Veamos el documento con un poco más de detalle:

```
<?xml version="1.0" encoding="UTF-8"?>
<MobMet:Application xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http
  <Screens xsi:type="MobMet:MenuScreen" name="MenuScreen" tittle="Menu Principal">
    <MenuElements class="" name="ms1" value="Atracciones" TargetScreen="//@Screens.3">
      <Icon name="imgElement" source="logo2_3.jpg" EReference0="//@Screens.0/@MenuElemer
    </MenuElements>
    <MenuElements id="1" name="ms2" value="Espectáculos" TargetScreen="//@Screens.7"/>
    <MenuElements id="2" name="ms3" value="Alojamientos" TargetScreen="//@Screens.20"/>
    <MenuElements id="3" name="ex" value="Salir" TargetScreen="//@Screens.2"/>
  </Screens>



 <Screens xsi:type="MobMet:StartScreen" name="StartSplashScreen" tittle="Inicio" transiti
   <Images class="" name="imgStart" source="main.png" EReference0="//@Screens.1/@Images.0
 </Screens>
 <Screens xsi:type="MobMet:FinishScreen" name="EndSplashScreen">
   <Images id="1" class="" name="imgFinish" source="portaFin.png" EReference0="//@Screens
 </Screens>
 <Screens xsi:type="MobMet:MenuScreen" name="Areas_Atracciones" tittle="Áreas">
   <MenuElements class="" name="China" value="China" TargetScreen="//@Screens.4">
     <Icon name="imgElem" source="point.gif" EReference0="//@Screens.3/@MenuElements.0/@I
   </MenuElements>
   <MenuElements id="1" name="Far West" value="Far West" TargetScreen="//@Screens.23"/>
   <MenuElements id="4" name="Polynesia" value="Polynesia" TargetScreen="//@Screens.24"/>
   <MenuElements id="3" name="México" value="México" TargetScreen="//@Screens.25"/>
   <MenuElements id="2" name="Mediterrània" value="Mediterrània" TargetScreen="//@Screens
   <MenuElements id="5" name="volver" value="Volver" TargetScreen="//@Screens.0"/>
 </Screens>
```

Figura 55: default.mobilemetamodel de Portaventura Guide

De esta manera, se definen los distintos elementos que contiene la aplicación. Por ejemplo, tenemos un elemento *Screens*, que contiene varios *MenuElements*. Cada uno de estos elementos, se corresponde con una entrada de menú, siendo China, Far West, Polynesia, México, Mediterrània y Volver.

En el caso de imágenes, se especifica por ejemplo la propiedad *source*, que indica la ruta de la imagen a mostrar.

¿Cómo se enlazan las distintas pantallas? Es decir, ¿Cómo se sabe a la hora de generar código, que por ejemplo la opción de menú China, enlaza con la pantalla de información de esa área? La respuesta es, por posiciones.

Los elementos tienen un elemento *TargetScreen*, cuyo contenido indica a la hora de generar, en qué parte de fichero *default.mobileMetamodel* se encuentra la pantalla que enlaza.

Por ejemplo: La opción de menú México, hace referencia a //@Screens.25. Esto se traduce en el elemento 25 de tipo Screens que se encuentre en el fichero.

De esta manera se va procesando el fichero y se va generando el código correspondiente.

5.3.3. Conversión

En el anterior apartado, hemos comentado que los enlaces entre pantallas van por posiciones. Esta característica ha sido una de las más difíciles de emular con la aplicación MovilTurismo…pero vayamos por partes.

El primer reto al que nos hemos enfrentado, ha sido el componente editor de PrimeFaces. Como hemos comentado anteriormente, este componente es capaz de "digerir" una gran variedad de código HTML.

Según las características que debía poseer la aplicación y acorde al metamodelo existente, sólo se han tenido en cuenta cierto número de elementos HTML, tales como imágenes, links, listas, texto y texto en negrita.

Así pues, el primer paso fue extraer el código HTML generado por el editor y e ir tratando las distintas etiquetas para poder construir el nuevo fichero.

El editor HTML de PrimeFaces, posee el siguiente aspecto:

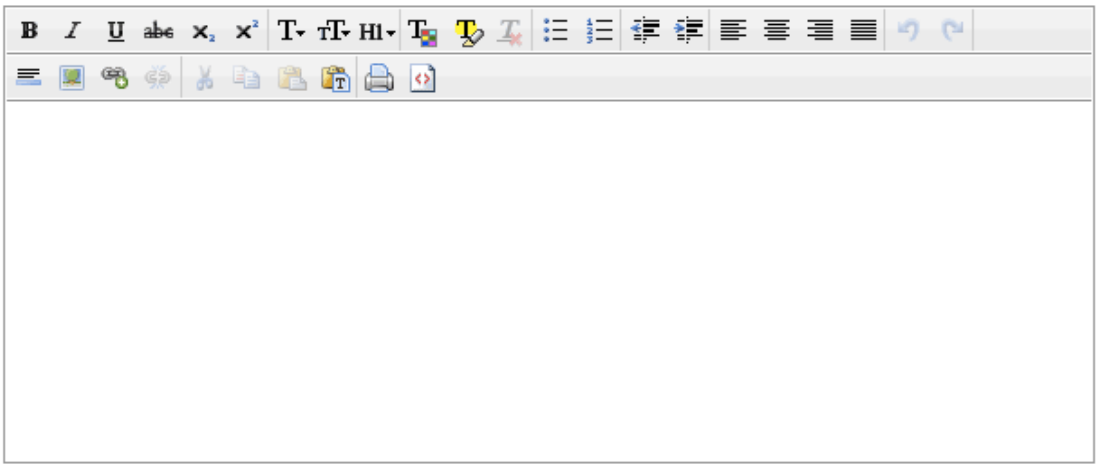


Figura 56: Componente Editor de PrimeFaces

Cómo puede observarse existe un gran número de botones para añadir componentes (a parte del propio editor). En MovilTurismo, el número de botones se ha reducido para adecuarse al metamodelo.

Para que la aplicación fuese capaz de procesar algunos elementos más, fue necesario ampliar el metamodelo. Los cambios han consistido en añadir al elemento *Text*, los siguientes atributos y son visibles en el Anexo 2:

<eStructuralFeatures xsi:type="ecore:EAttribute" name="**bold**" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>

<eStructuralFeatures xsi:type="ecore:EAttribute" name="**style**" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>

<eStructuralFeatures xsi:type="ecore:EAttribute" name="**stylebold**" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>

<eStructuralFeatures xsi:type="ecore:EAttribute" name="**ref**" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>

<eStructuralFeatures xsi:type="ecore:EAttribute" name="**mostrar**" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>

Esta es la explicación para cada uno de ellos:

- **bold**: Indica el texto que irá en negrita
- **style**: Indica el estilo del texto
- **stylebold**: Indica el estilo del texto en negrita
- **ref**: Indica la url del link
- **mostrar**: Indica si es un link

En un primer momento, se intentó procesar el código HTML de manera manual, pero finalmente se usó la librería *commons-io* [25] para este cometido. El funcionamiento es el siguiente:

Teniendo una cadena como esta: **<STRONG>Título</STRONG><P>Esto es el contenido</P>**, es necesario procesar la cadena indicándole cual será el inicio y cuál será el fin. Si se establece como principio y fin **<STRONG>** y **</STRONG>,** la ejecución nos devolverá *Título*

Siguiendo este mecanismo he ido extrayendo la información necesaria. Los componentes que se llegan a procesar en la actualidad son:

<STRONG>: Negrita

<UL><LI>: Listas ordenadas

<P>: Párrafo

<IMG>: Imagen

<A HREF>: Enlaces

El contenido se transforma en propiedades de los elementos del XML o incluso en nuevos elementos.

Como hemos visto en el esquema del inicio de este apartado, la información extraída va volcándose en el fichero *tesis.mobilemetamodel* guardando el formato esperado. Es muy importante no olvidarse de ninguna propiedad y ni dejar ninguna sin valor, puesto que ello se traduce en un error de ejecución.

MovilTurismo cuenta con distintos controles para que esto no ocurra. La pestaña de generación de código no nos dejará generar nada si por ejemplo, existe alguna pantalla sin contenido o si no existe un Splash definido, etc… (leer el punto 7: Manual de Usuario)

5.4. Generando documentos XMI con JSF

Llegados a este punto, hemos revisado como se realiza el parseo con el editor HTML de PrimeFaces, pero eso no es todo. El siguiente componente "estrella" es el componente *Tree*. Dicho componente es esencial para establecer la jerarquía de la aplicación, las conexiones entre las diferentes pantallas y los menús. Por supuesto, todo esto debe volcarse en el fichero *tesis.mobilemetamodel*

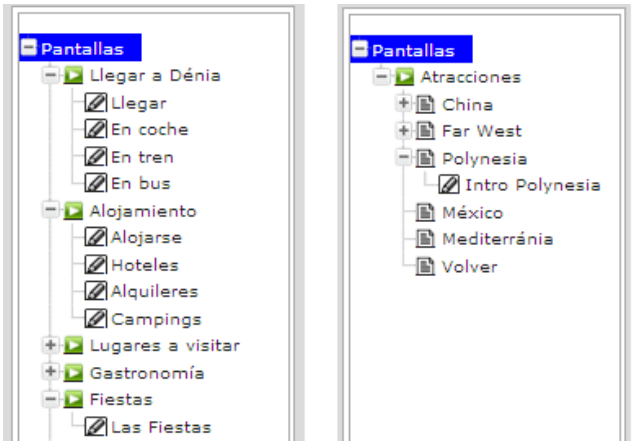


Figura 57:Ejemplos de Tree

La creación de menús en la aplicación móvil es una decisión que controla totalmente el usuario. Por ello, nunca podemos estar seguros de qué estructura va a seguir cada aplicación que el usuario genere. En otras palabras, resulta imposible acotar una estructura básica.

Por ello, en MovilTurismo no se ha especificado ninguna restricción: Todo será a juicio del propio usuario.

Esta libertad se traduce en un mayor control a la hora de generar la aplicación. En la figura anterior, el árbol de la izquierda nos muestra una estructura en la que el usuario ha creado un menú principal y ha colgado pantallas de él. El árbol de la derecha, muestra además, cómo de una de las opciones del menú vuelve a crearse otro menú.

Esto ha obligado a almacenar en base de datos información adicional para poder dar esta flexibilidad. La tabla auxiliar *wm_menu* es la encargada de hacer esto posible. Cada vez que el usuario crea una nueva opción de menú, se almacena en esta tabla un nuevo registro. La nueva entrada de menú, harán referencia al nuevo registro. Así, a la hora de generar el fichero, es posible agrupar las pantallas de este tipo en los menús correspondientes.

Cada vez que se crea una nueva pantalla del tipo que sea, se almacena un nuevo registro en la tabla *wm_screen*. En esta tabla, nos podemos encontrar registros de diferente tipo:

- MEN: Registros que hacen referencia a entradas de menú
- EDI: Registros que hacen referencia a pantallas de contenido
- STA: Registro que hacen referencia al menú principal

La gestión de plantillas es algo menos compleja. También se gestiona con un componente árbol, pero no se permite tanto nivel de jerarquía como en las pantallas.

A nivel de generación, la aplicación recorrerá las tablas wm_template y wm_submenu, para realizar el volcado de datos en el fichero *tesis.mobilemetamodel*. Para más información sobre las tablas del modelo relacional, consultar el apartado 4.6.

Por último de cara a la conversión comentaremos cómo se realiza el enlace entre pantallas. En el XML del fichero gráfico, tal y como hemos visto en el punto 4.3.1, las pantallas se van enlazando según la posición o el número de elemento que se encuentra en el fichero.

Conforme el usuario va creando su nueva aplicación, es vital guardarse qué posición va a ocupar cada una de las pantallas que está creando. Esto puede parecer trivial, pero llega a complicarse cuando el usuario empieza a crear menús y cuando entra el juego el borrado de algún nodo del árbol.

Para ello se creó la columna *contador* presente en algunas tablas: La idea era sencilla, cuando el usuario cree una nueva pantalla, yo incremento el contador y lo almaceno en base de datos para saber en qué posición del fichero debo volcar la pantalla.

Sin embargo, este planteamiento no funcionaba correctamente en el caso de los menús. Veámoslo con un ejemplo según el funcionamiento de la aplicación:

Un usuario creó una nueva entrada de menú nueva llamada *China*. Cómo es el primer elemento del nuevo menú y no hay menú asociado, se crea en la tabla wm_menu una entrada que indica que se ha generado un menú: lo llamaremos *Áreas*.

Seguidamente, introducimos otra entrada de menú, a la que llamaremos *Polynesia*. Puesto que queremos que sea una opción de Áreas, debe colgar del mismo nodo que China. Como Áreas ya estaba creado, no se crea una nueva entrada en wm_menu.

Siguiendo el planteamiento, se apuntaría como contador 1 para *Áreas*, 2 para *China* y 3 para *Polynesia* y se volcarían en ese orden. Sin embargo si echamos un vistazo al *default.mobilemetamodel* (Figura 45), podemos ver como los elementos *China* y *Polynesia*, no son pantallas, si no elementos MenuElements **incluidos** en la pantalla (Screens) *Áreas* y que por lo tanto no deberían aumentar su contador de posición, si no **tener el mismo que el elemento Screens que las contiene**.

Teniendo en cuenta estas consideraciones, tendríamos el siguiente algoritmo de ejecución para el volcado de pantallas (sin contar plantillas):

Contador = Máximo contador en base de datos

ContadorAuxiliar = 0

Mientras ContadorAuxiliar < Contador

Recorremo tablas wm_screen y buscamos **los** registros con ese contador

Si el registro es de tipo **MEN** o **STA**

Vamos a la tabla wm_menu y miramos el registro que contiene el mismo contador

Volcamos a tesis.mobilemetamodel un nuevo menú en la posicion ContadorAuxiliar

Volcamos a tesis.mobilemetamodel los MenuElement dentro del menú

Si el registro es de tipo **EDI**

Volcamos el elemento Screen con las características del registro de wm_screen encontrado a través del parseo del HTML del editor de PrimeFaces.

ContadorAuxiliar ++

Fin Mientras

El elemento contador, es clave para la generación y permite saber la posición del elemento Screens en el fichero tesis.mobilemetamodel.

Las plantillas van al final del fichero gráfico, con lo que están en una tabla aparte wm_template. Para realizar el volcado se recorrer los registros según la columna contador (que se va incrementando con cada inserción) y se realizan cruces con la tabla wm_submenu, para volcar los enlaces disponibles desde las plantillas.

5.5. Datos de Soporte

Vemos a continuación el listado de tablas sobre las que se apoya la aplicación, así como una descripción de cada una de ellas y de los campos que las forman:

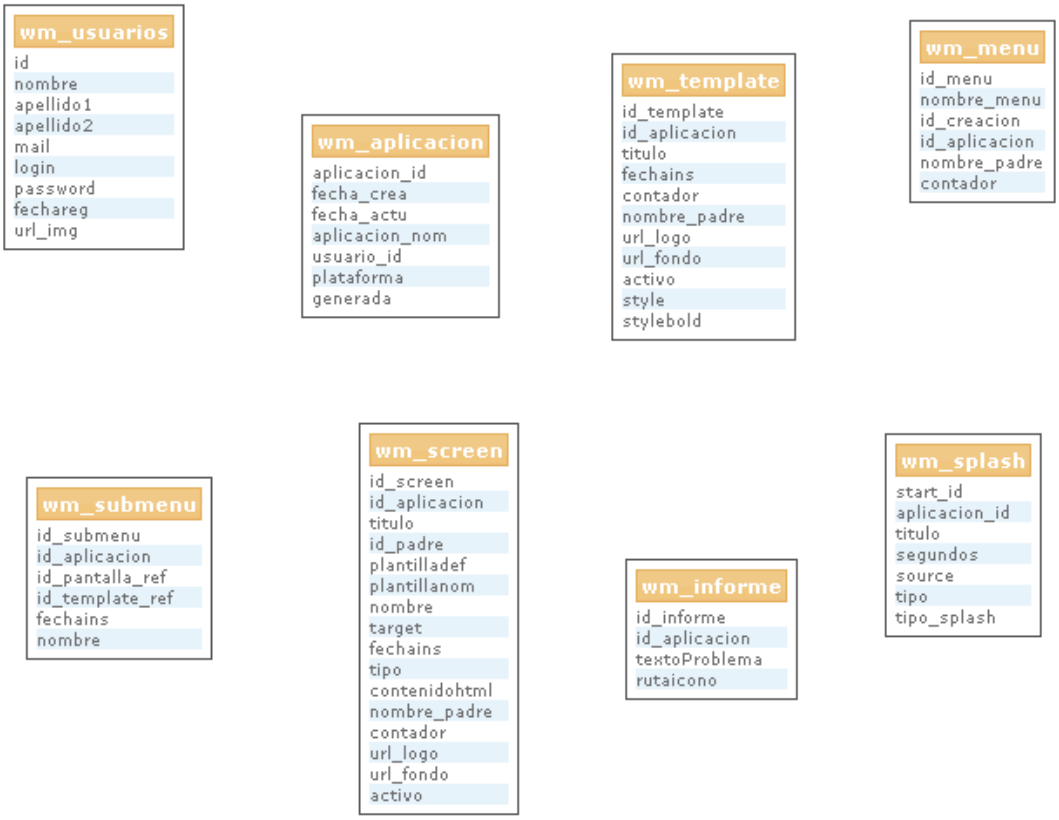


Figura 58: Tablas de la aplicación

5.5.1.1. Inventario de Tablas

TABLA	DESCRIPCIÓN	TIPO
wm_usuarios	Esta tabla contiene los usuarios de la aplicación	O
wm_aplicacion	Contiene cada una de las aplicaciones creadas	O
wm_template	Contiene información sobre las plantillas creadas	O
wm_menu	Esta tabla lleva la cuenta de cuentos menús se han creado y qué posición ocupan. Es muy importante para generar el fichero que simula el editor gráfico.	O
wm_submenu	Esta tabla contiene información acerca de los submenús de las plantillas	O

TABLA	DESCRIPCIÓN	TIPO
wm_screen	Contiene información sobre cada una de las tablas creadas	O
wm_informe	Esta tabla almacena si ha habido algún problema a la hora de generar el código.	O
wm_splash	Contiene información de las pantallas splash inicales y finales.	O

Tabla 3: Inventario de tablas

P: Tabla paramétrica

O: Tabla Operativa

5.5.1.2. Tabla wm_usuarios

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
id	INT(10)	Identificador del usuario	S	N
nombre	VARCHAR(100)	Nombre del usuario	N	N
apellido1	VARCHAR(100)	Primer apellido del usuario	N	N
apellido2	VARCHAR(100)	Segundo apellido del usuario	N	N
mail	VARCHAR(100)	Dirección de correo electrónico	N	N
login	VARCHAR(50)	Login	N	N
password	VARCHAR(50)	Contraseña	N	N
fechareg	DATE	Fecha de registro en el sistema	N	N
url_img	VARCHAR(500)	Url de la imagen del perfil del usuario	N	N

Tabla 4: Contenido tabla wm_usuarios

5.5.1.3. Tabla wm_aplicacion

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
aplicación_id	INT(10)	Identificador de la aplicación	S	N
fecha_crea	DATE	Fecha de creación de la aplicación	N	N
fecha_actu	DATE	Fecha de actualización de la aplicación	N	N
aplicación_nom	VARCHAR(150)	Nombre de la aplicación	N	N
usuario_id	INT(10)	Identificador del usuario creador	N	S
plataforma	VARCHAR(10)	Plataforma de la aplicación a generar	N	N

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
generada	VARCHAR(200)	Indicador de si la aplicación ya ha sido generada	N	N

Tabla 5: Contenido tabla wm_aplicacion

5.5.1.4. Tabla wm_template

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
id_template	INT(11)	Identificador de la plantilla	S	N
id_aplicacion	INT(11)	Identificador de la aplicación	N	N
titulo	VARCHAR(50)	Título de la aplicación	N	N
fechains	DATETIME	Fecha de inserción	N	N
contador	INT(11)	Indica la posición de volcado en el fichero generador	N	N
nombre_padre	VARCHAR(50)	Indica el nombre del padre en el árbol	N	N
url_logo	VARCHAR(500)	Url de la imagen perteneciente al logo	N	N
url_fondo	VARCHAR(500)	Url de la imagen perteneciente al fondo	N	N
activo	VARCHAR(1)	Indica si la plantilla está activa	N	N
style	VARCHAR(200)	Estilo elegido para la plantilla	N	N
stylebold	VARCHAR(200)	Estilo negrita elegido para la plantilla	N	N

Tabla 6: Contenido Tabla wm_template

5.5.1.5. Tabla wm_menu

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
id_menu	INT(11)	Identificador del menú	S	N
nombre_menu	VARCHAR(150)	Nombre del menú	N	N
id_creacion	INT(11)	Identificador creación	N	N
id_aplicacion	INT(11)	Identificador de la aplicación	N	N
nombre_padre	VARCHAR(150)	Nombre del padre en el árbol de nodos	N	N
contador	INT(11)	Posición del menú en el fichero de generación	N	N

Tabla 7: Contenido Tabla wm_menu

5.5.1.6. Tabla wm_submenu

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
id_submenu	INT(11)	Identificador del submenú	S	N
id_aplicacion	INT(11)	Identificador de la aplicación	N	N
id_pantalla_ref	INT(11)	Identificador de la pantalla de referencia	N	N
id_template_ref	INT(11)	Identificador de la plantilla de referencia	N	N
fechains	DATETIME	Fecha de inserción	N	N
nombre	VARCHAR(50)	Nombre del submenú	N	N

Tabla 8: Contenido Tabla wm_submenu

5.5.1.7. Tabla wm_screen

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
id_screen	INT(11)	Identificador de la pantalla	S	N
id_aplicacion	INT(11)	Identificador de la aplicación	N	S
titulo	VARCHAR(100)	Título de la pantalla	N	S
Id_padre	INT(11)	Identificador del padre	N	S
plantilladef	VARCHAR(1)	Indica si es la plantilla por defecto	N	N
plantillanom	VARCHAR(150)	Nombre de la plantilla	N	N
nombre	VARCHAR(100)	Nombre de la pantalla	N	N
target	VARCHAR(100)	Pantalla destino	N	N
fechains	DATETIME	Fecha de inserción	N	N
tipo	VARCHAR(3)	Tipo de pantalla	N	N
contenidohtml	VARCHAR(15000)	Contenido html del editor	N	N
nombre_padre	VARCHAR(150)	Nombre del padre de la pantalla	N	N
contador	INT(11)	Posición en el fichero gráfico	N	N
url_logo	VARCHAR(500)	Url del logo de la pantalla	N	N
url_fondo	VARCHAR(500)	Url del fondo de la pantalla	N	N
activo	VARCHAR(5)	Indica si la pantalla está activa	N	N

Tabla 9: Contenido Tabla wm_screen

5.5.1.8. Tabla wm_informe

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
id_informe	INT(11)	Identificador del informe	S	N
id_aplicacion	INT(11)	Identificador de la aplicación	N	S
textoProblema	VARCHAR(500)	Información del problema	N	N
rutaicono	VARCHAR(150)	Ruta del icono del problema	N	N

Tabla 10: Contenido Tabla wm_informe

5.5.1.9. Tabla wm_splash

CAMPO	TIPO	DESCRIPCIÓN	PK	FK
start_id	INT(10)	Identificador del start / finish splash	S	N
aplicación_id	INT(10)	Identificador de la aplicación	N	S
titulo	VARCHAR(150)	Título del splash	N	N
segundos	INT(10)	Segundos de duración en pantalla	N	N
source	VARCHAR(500)	Url de la imagen	N	N
tipo	VARCHAR(20)	Tipo de la imagen	N	N
tipo_splash	VARCHAR(3)	Distingue si es start o finish	N	N

Tabla 11: Contenido Tabla wm_splash

5.6. Aplicaciones desarrolladas con MovilTurismo

Existen diferencias significativas a la hora de observar el resultado de generar código que dependen de la plataforma que se ha elegido para la nueva aplicación así como de la versión que estamos ejecutando. Por ello, hemos separada el presente apartado en 3 puntos: *GMF*, *Blackberry* y *HTML5*

5.6.1.1. MovilTurismo / GMF

Hemos comentado anteriormente, que en la versión anterior cuando se generan las clases con Xpand2, se vuelca el resultado a la carpeta *src-gen* del proyecto OAW. Las clases Java generadas (así como las imágenes) deberán importarse a un proyecto de tipo *Blackberry* en una suite que cuente con los plugins correspondientes, por ejemplo Eclipse.

Una vez hecho esto, obtuvimos la versión final del proyecto:

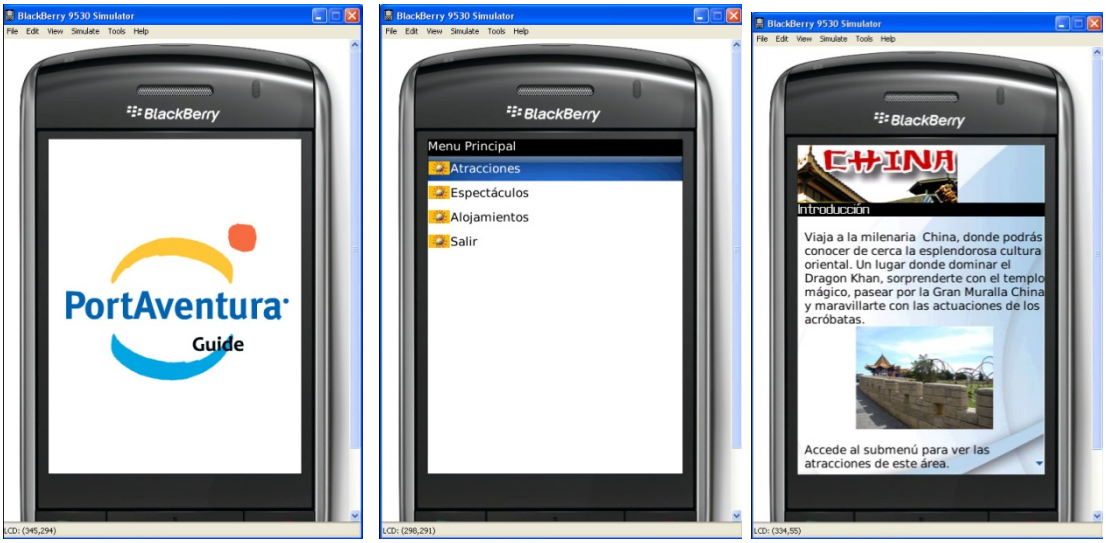


Figura 59: Versión final Portaventura Guide

5.6.1.2. Aplicación Nativa Blackberry desarrollada

Una vez el código se ha generado, éste se vuelca en la carpeta TMP del equipo, siguiendo la siguiente estructura de directorios:

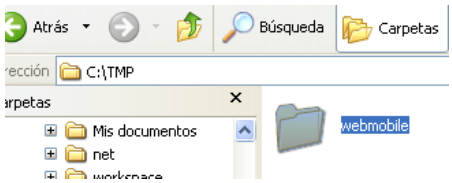


Figura 60: Estructura directorios Blackberry (I)

En primer lugar se crea una carpeta webmobile, que será la carpeta contenedora del código generado. Dentro de esta carpeta nos encontramos con tres elementos:

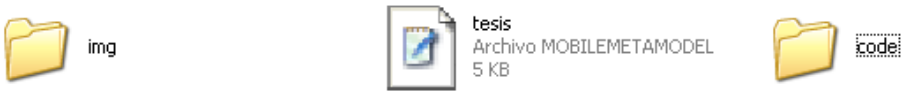


Figura 61: Estructura directorios Blackberry (II)

img: es la carpeta que contiene las imágenes de la aplicación. La sintaxis del código para Blackberry, no permite especificar como path de imágenes urls. Por ello, cuando se genera la aplicación, se vuelcan todas las imágenes usadas en esta carpeta. Por otro lado, **tesis.mobilemetamodel**, es el nuevo fichero gráfico que se ha creado durante la generación. En él, se encuentra la información de la aplicación en formato XML acorde con el metamodelo. Por último la carpeta **code**, contiene todas las clases Java que se han generado.

Una vez tenemos todo listo, el usuario deberá coger el código y las imágenes y compilar el proyecto en la suite correspondiente (por ejemplo Eclipse) para poder probarlo, y, finalmente, subirlo a la tienda de aplicaciones de Blackberry si lo estima oportuno. Este parte del proceso es idéntica a la versión GMF.

5.6.1.3. HTML5

A diferencia de la generación orientada a la plataforma Blackberry, cuando se genera el código desde MovilTurismo, no se vuelca el resultado en el fichero *tmp* del ordenador, si no que lo coloca directamente en la carpeta del servidor.

Esto supone que, una vez generada la aplicación estará inmediatamente disponible para el uso y disfrute del usuario y no deberá esperar a colgarla en una tienda de aplicaciones, que la validen, etc... En resumen, el usuario podrá usar la aplicación desde el mismo momento que se genere.

La estructura de directorios generada es la siguiente:

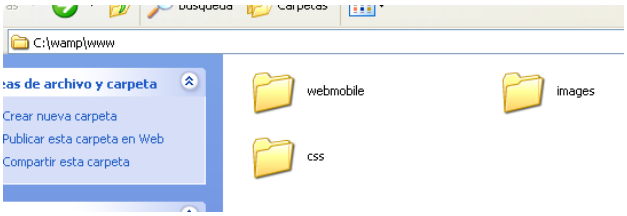


Figura 62: Estructura directorios HTML5 (I)

En la carpeta raíz del servidor, tendremos disponible una carpeta **images** y una carpeta **css**. Estas dos carpetas no se crean durante la generación, si no que permanecen en el servidor siempre.

La carpeta *images*, contiene las imágenes que son comunes para todas las aplicaciones generadas. Esto es, el fondo de pantalla, la cabecera, botones, iconos, etc... y la carpeta *css* incluye los estilos css que se usarán en todas las aplicaciones.

Al igual que en el anterior caso, la generación provoca la creación de la carpeta *webmobile* que será la carpeta contenedora del código generado.

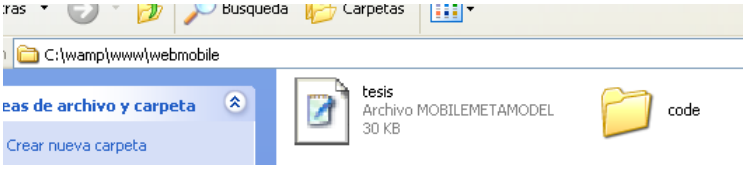


Figura 63: Estructura directorios HTML5 (II)

Dentro de este directorio entontramos el fichero gráfico **tesis.mobilemetamodel** y el directorio **code**. En este caso, dicho directorio contiene los archivos HTML generados y adicionalmente un directorio **img**, donde, de la misma manera que para Blackberry, se almacenan las imágenes usadas en la aplicación

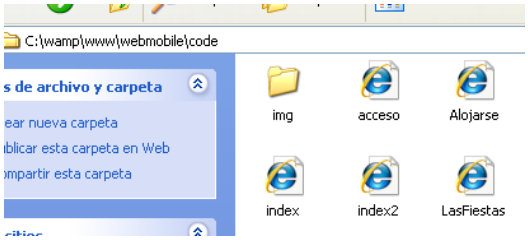


Figura 64: Estructura directorios HTML5 (III)

Llegados a este punto, el usuario no tiene más que ejecutar la aplicación en un navegador compatible con HTML5 y disfrutar del resultado obtenido.

5.7. Consideraciones finales

En anteriores apartados, hemos visto cómo se abordó el tema de la generación con las 2 versiones comentadas: GMF y JSF, así como las tecnologías que se han empleado.

A nivel de tecnología, las diferencias son significativas. Veámoslo en los siguientes esquemas:

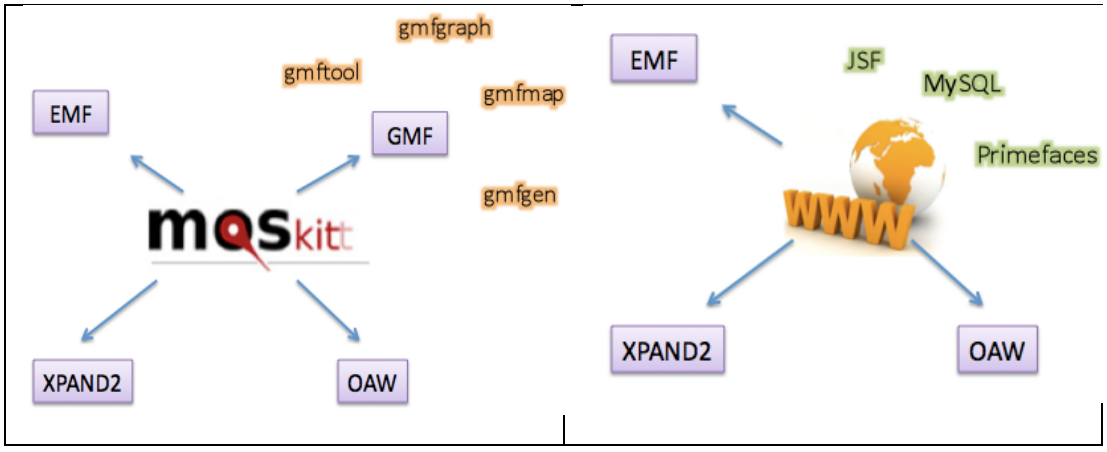


Figura 65: Esquema solución previa y actual

En el esquema de la parte de la izquierda, podemos ver como se usaba GMF para diseñar la propia aplicación. Esto se hace desde la suite MOSKitt, que permite adicionalmente (y tras la instalación de los plugins necesarios) realizar las tareas de conversión de código con las plantillas definidas en XPAND2 y a través de un proyecto OAW.

En el esquema de la parte derecha, podemos ver como en la solución actual, la parte de GMF ha desaparecido explícitamente, y se ha sustituido por tecnologías que han permitido crear la aplicación MovilTurismo, siendo ésta la encargada de ayudar al usuario a crear su aplicación.

Las tareas relativas a la conversión con el proyecto OAW y las plantillas en XPAND2, también recaen sobre la plataforma Web, ya que es capaz de usar esa tecnología en su ejecución.

¿Por qué este cambio? La primera versión, es eficiente, pero sin embargo GMF no es nada intuitivo para un usuario final ya que requiere de un proceso de aprendizaje ciertamente complejo.

Sin embargo, la solución propuesta, permite a un usuario sin conocimientos usar la misma tecnología ,pero esta vez a través de una interfaz más amigable que le facilite el proceso de creación.

6.Caso de Estudio: Guía Turística de Dénia

En el presente apartado, vamos a ver a groso modo la creación de la aplicación "Guía Turística de Dénia.

6.1. Descripción del sistema

El presente caso de estudio tratar de emular la creación de una guía turística, en este caso, para la ciudad de Dénia, en la que se recojan los principales lugares turísticos para que los turistas puedan estas informados en todo momento de qué pueden visitar, dónde se encuentra, dónde comer, etc.…

El tipo de plataforma para realizar este caso de uso ha sido HTML5 y estéticamente tiene la apariencia de una aplicación para iPhone.

6.2. Especificación

Para empezar a crear la aplicación, lo primero que hay que hacer una vez se está validado en el sistema y se ha elegido el tipo de aplicación a crear, es crear el Start Splash.

En este apartado vamos a ver en reglas generales como se ha definido la aplicación del caso se uso. Para ver en detalle el funcionamiento de todos los componentes, es imprescindible leer primero el apartado 7: Manual de usuario.

La pestaña de Start Splash es la encargada de crear la pantalla de presentación de la aplicación. A través de un sencillo formulario el usuario puede especificar los valores que se usarán en la creación de la pantalla, tal y como se observa a continuación.

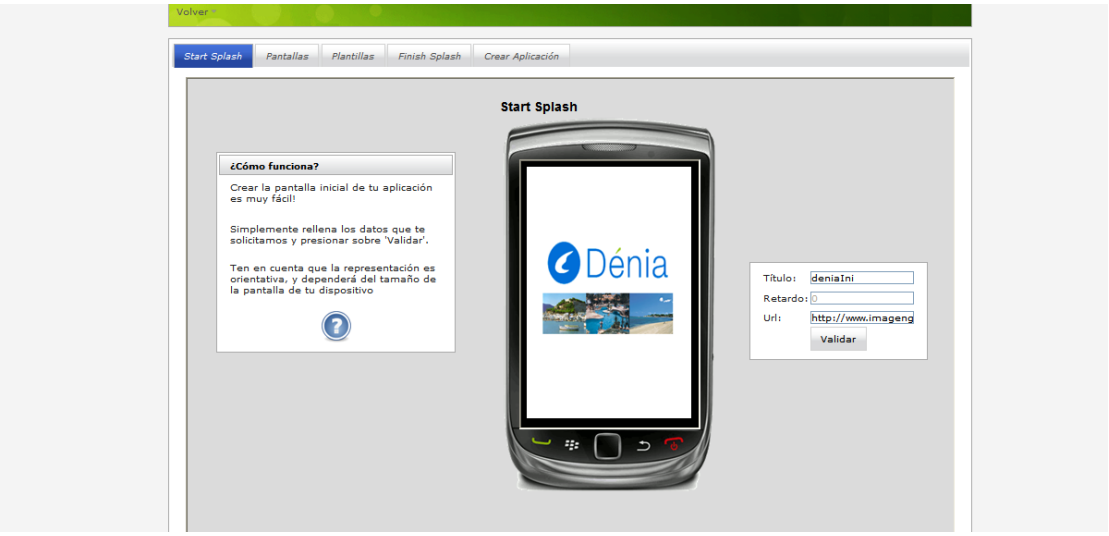


Figura 66: Caso de Estudio - Start Splash

En la pestaña "Pantallas" se especifica la jerarquía de las pantallas de la aplicación, así como algunas relaciones. Las siguientes figuras nos muestran parte de las pantallas creadas. A la parte izquierda, se distingue el componente árbol que indica dicha jerarquía. Puede observarse por ejemplo, que se ha creado el siguiente menú principal:

- Llegar a Dénia
- Alojamiento
- Lugares a Visitar
- Gastronomía
- Fiestas
- Salir

Y que a partir de el menú, "Llegar a Dénia", se han creado las pantallas

- Llegar (cuyo contenido está en edición)
- En coche
- En tren
- En bus

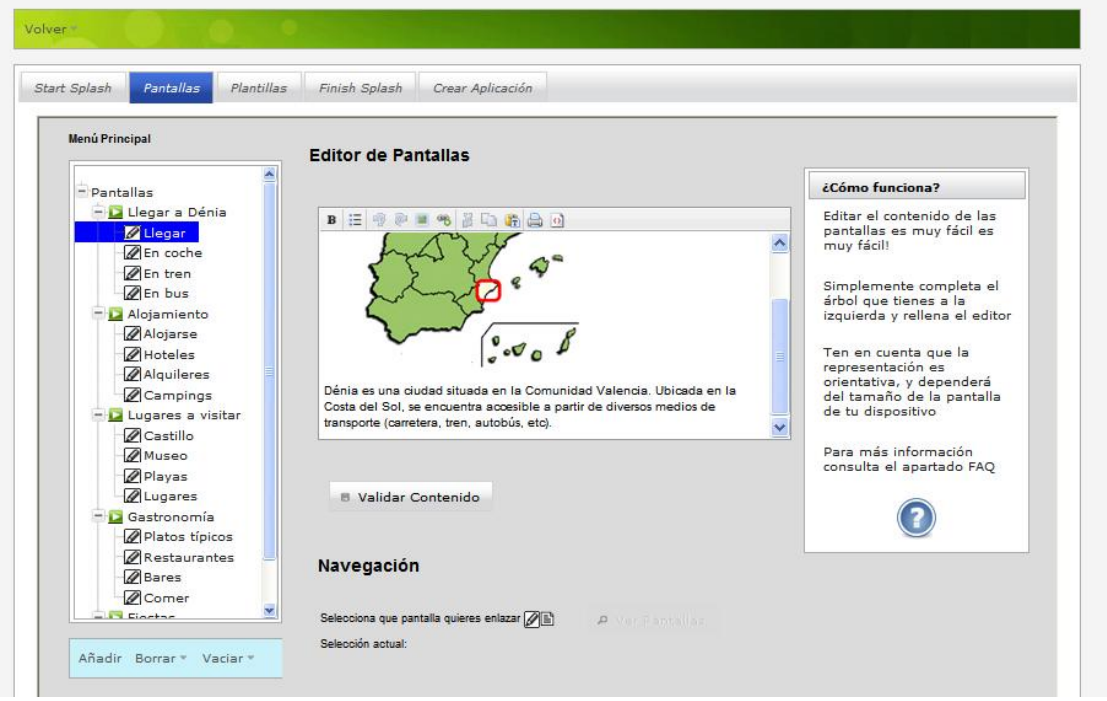


Figura 67: Caso de Estudio - Pantallas (I)

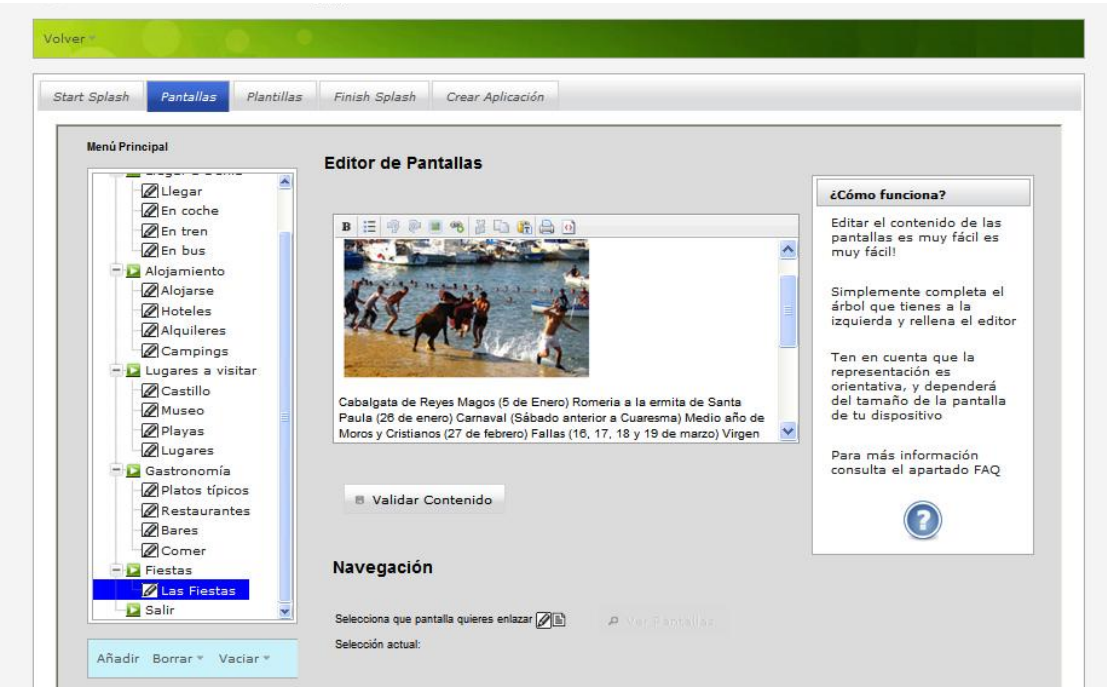


Figura 68: Caso de estudio - Pantallas (II)

En la pestaña Plantillas hemos establecido la apariencia de algunas de las pantallas. Por ejemplo, se ha establecido que para la plantilla "*Template_llegar*" (que usan las pantallas *En coche*, *En Tren* y *En bus*) el color del tipo de letra en negrita sea amarillo.

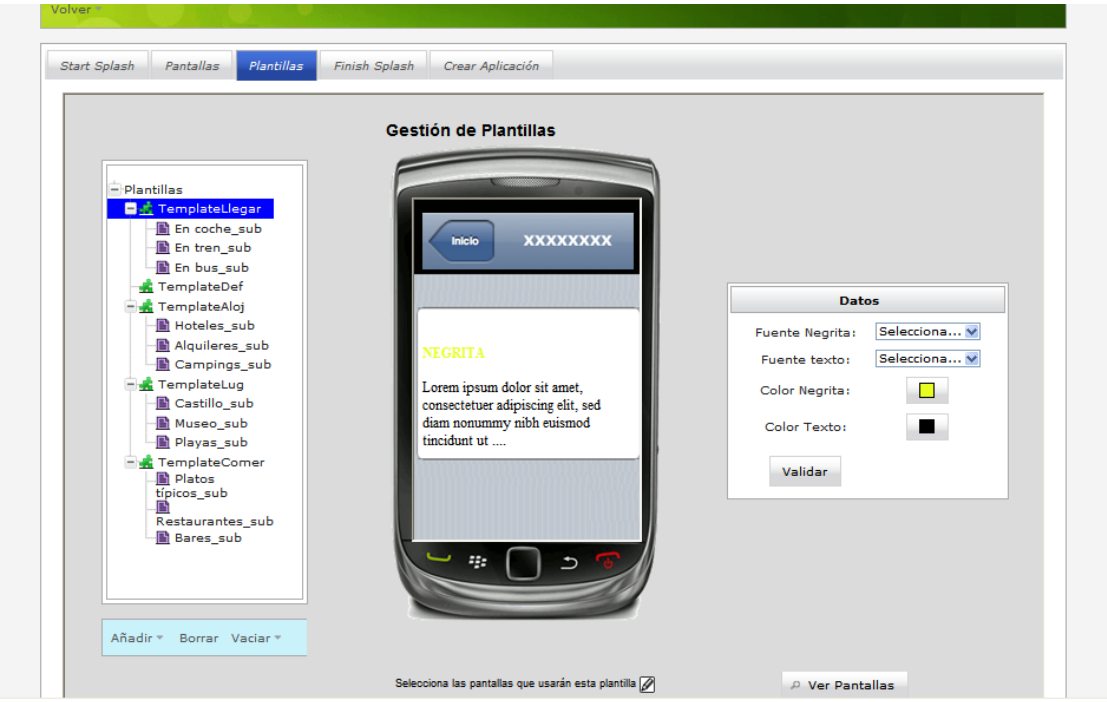


Figura 69: Caso de estudio: Plantillas

Siguiendo con la pestaña Finish Start, en ella se configura el equivalente a la pantalla de inicio de la aplicación, pero en este caso, se trata de la pantalla de "despedida"

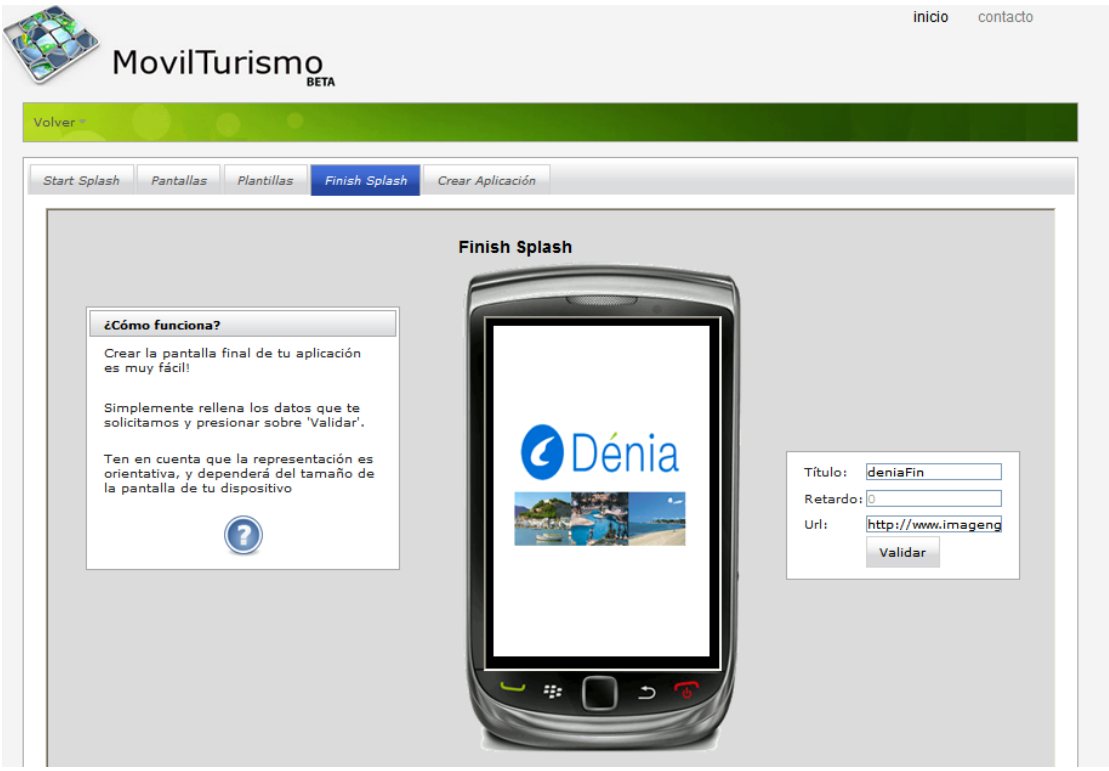


Figura 70: Caso de estudio - Finish Splash

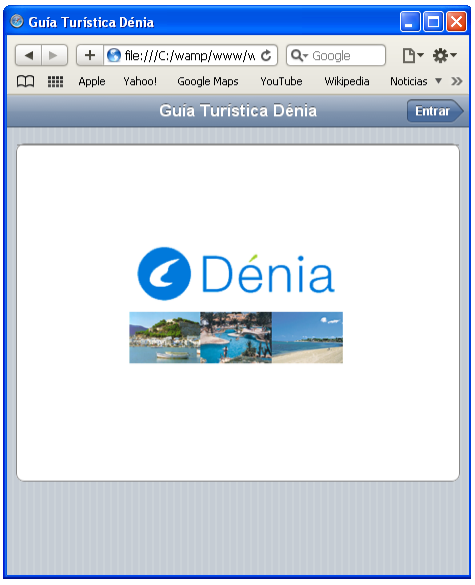
Por último, desde la pestaña “Crear Aplicación” lanzamos el mecanismo de generación de código y esperamos los resultados finales.

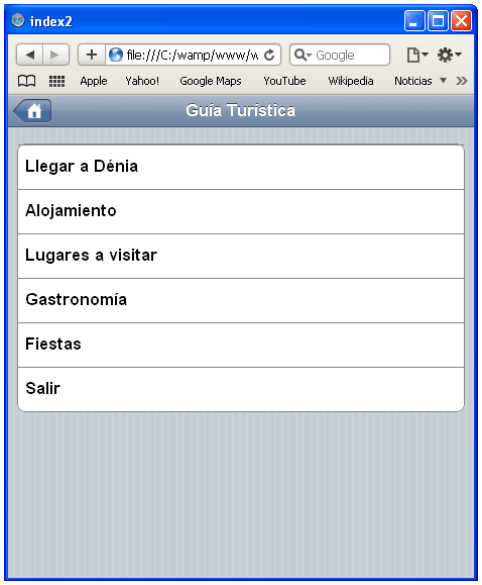
Figura 71: Caso de Uso - Crear Aplicación

6.3. Aplicación generada

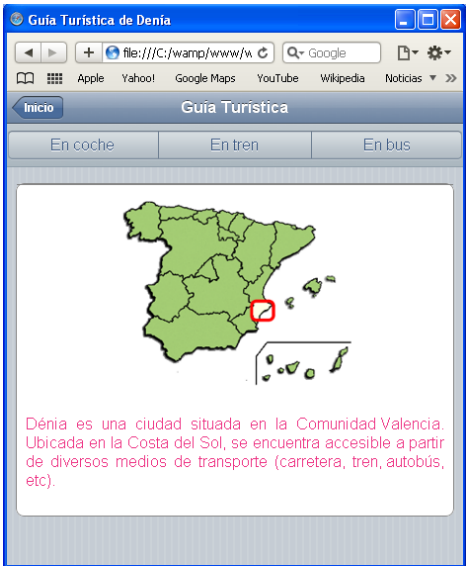
Una vez se ha establecido la jerarquía, contenido y relaciones entre las pantallas de la aplicación y se ha generado el código tras el proceso correspondiente, este es el aspecto de la aplicación generada:

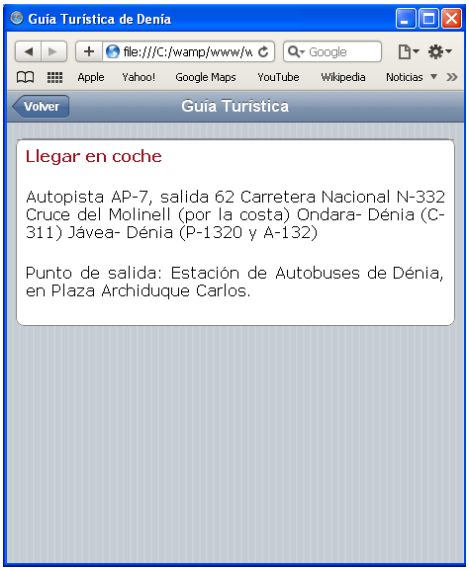
Pantalla de Inicio y Menú Principal

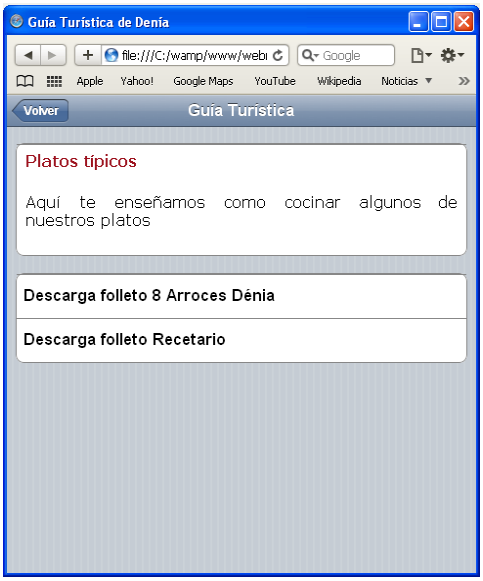


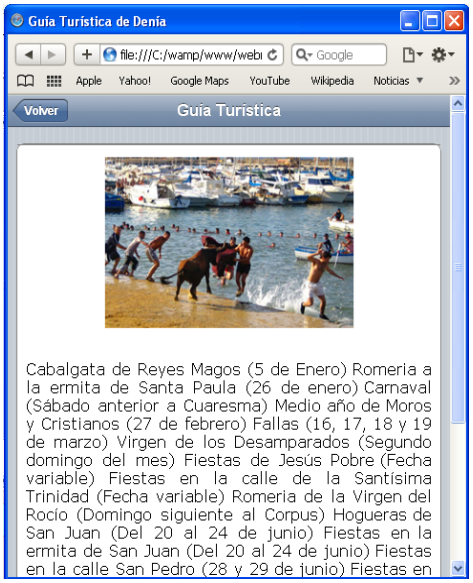


Ejemplos de Pantallas






Guía Turística de Denia
file:///C:/wamp/www/webi
Google
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
Noticias
Volver
Guía Turística
Platos típicos
Aquí te enseñamos como cocinar algunos de nuestros platos
Descarga folleto 8 Arroces Dénia
Descarga folleto Recetario


Guía Turística de Denia
file:///C:/wamp/www/webi
Google
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
Noticias
Volver
Guía Turística
Cabalgata de Reyes Magos (5 de Enero) Romeria a la ermita de Santa Paula (26 de enero) Carnaval (Sábado anterior a Cuaresma) Medio año de Moros y Cristianos (27 de febrero) Fallas (16, 17, 18 y 19 de marzo) Virgen de los Desamparados (Segundo domingo del mes) Fiestas de Jesús Pobre (Fecha variable) Fiestas en la calle de la Santísima Trinidad (Fecha variable) Romeria de la Virgen del Rocío (Domingo siguiente al Corpus) Hogueras de San Juan (Del 20 al 24 de junio) Fiestas en la ermita de San Juan (Del 20 al 24 de junio) Fiestas en la calle San Pedro (28 y 29 de junio) Fiestas en

7.Manual de Usuario

En este apartado veremos el funcionamiento de la aplicación con más detalle. Esta es la primera pantalla que el usuario se encontrará cada vez que entre a la aplicación:

Figura 72:Página principal MovilTurismo

En ella, se lista las principales características de la aplicación y permite al usuario identificarse, o si no dispone de identificación, realizar un nuevo registro.

Para realizarlo, el usuario deberá rellenar el formulario en pantalla y presionar sobre el botón "¡Registro!". Esta acción le llevará directamente a su página principal como usuario registrado: El Perfil de usuario

Si el usuario ya está dado de alta en el sistema, es necesario que introduzca los valores "Usuario" y "Password" y presione sobre el botón "Login" para acceder a su página de Perfil de Usuario

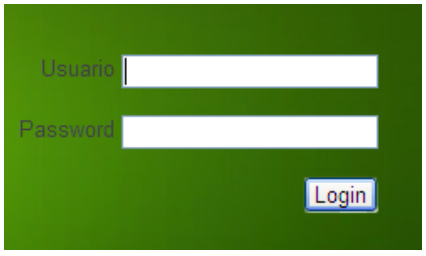


Figura 73: Formulario de identificación

Dicha página tiene el siguiente aspecto y en ella es donde se muestran los datos personales, así como las aplicaciones que tiene el usuario creadas.

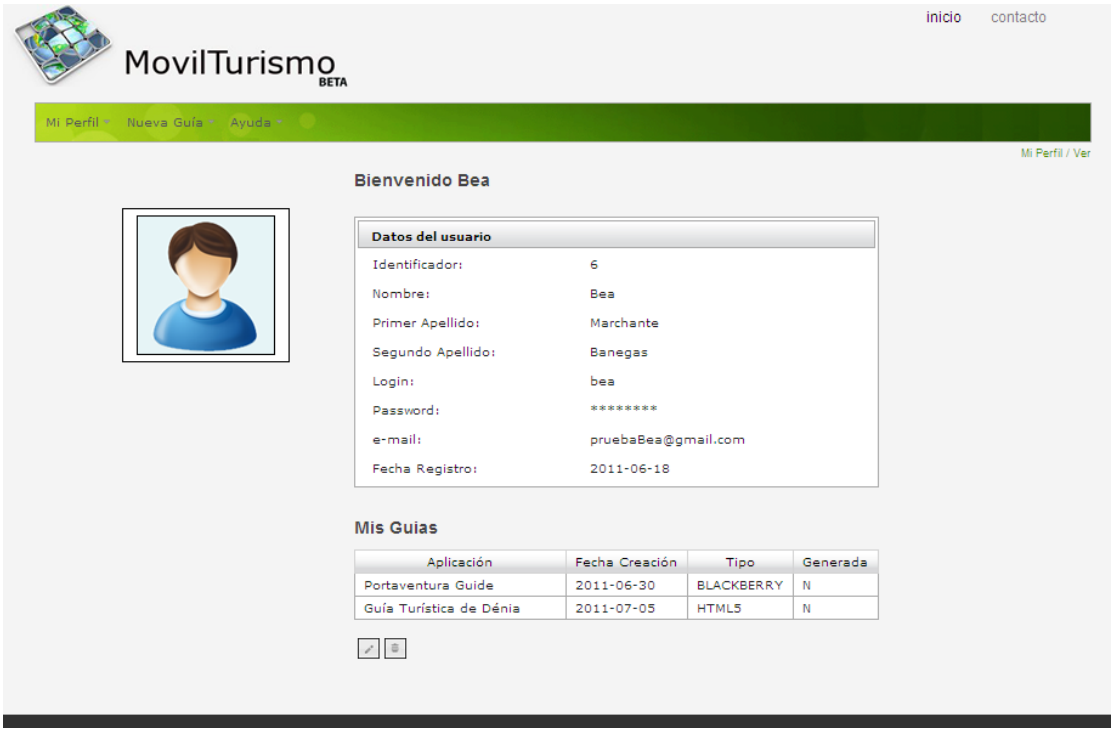


Figura 74: Perfil de usuario

La barra superior de la pantalla, muestra las opciones de menú disponibles: Si posicionamos el ratón sobre "*Mi Perfil*", aparecerá un menú desplegable con las opciones *Ver*, *Modificar* y *Cerrar Sesión*.

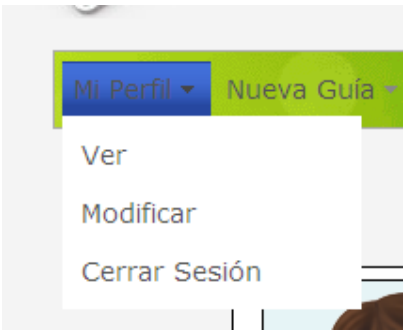


Figura 75: Opciones de "Mi Perfil"

La opción Ver es la opción visible en pantalla. Para modificar los datos personales del perfil, el usuario debe presionar sobre la opción *Modificar*. Esta acción le llevará a la pantalla de modificación, donde podrá cambiar sus datos personales rellenando el siguiente cuestionario:

Figura 76: Modificar Perfil

Para cambiar la imagen del perfil, basta con presionar sobre el botón *Cambiar Imagen* y aparecerá un diálogo, solicitando al usuario la url de la imagen, que quiere mostrar como perfil. Para que los cambios sean efectivos es necesario volver a iniciar sesión.

Volviendo al menú desplegable, el usuario podrá volver a la pantalla del perfil, seleccionando la opción *Ver* o podrá cerrar su sesión de usuario, desde la opción *Cerrar Sesión*.

Como hemos mencionado anteriormente, desde la pantalla del perfil es posible visualizar las aplicaciones que tiene creadas el usuario. Además, desde esta pantalla es donde se administran y donde podemos crearlas, modificarlas o borrarlas.

La tabla de la parte inferior de la pantalla, muestra una fila por cada una de las aplicaciones. Como información se muestra el nombre de la aplicación, la fecha de creación, el tipo y un indicador que nos dice si la aplicación ha sido generada o no.

Para continuar editando la aplicación, el usuario sólo debe seleccionar una fila de la tabla y pulsar sobre el botón de edición  . Debe seguir el mismo procedimiento para eliminar una aplicación, pero esta vez, seleccionando el botón de borrado  .

La ayuda de la aplicación se encuentra en el menú superior, donde el usuario accederá a una página con las preguntas más frecuentes acerca de MovilTurismo y desde donde podrá también, descargarse este manual de usuario.

Figura 77: Preguntas Frecuentes

Acabando con las opciones de menú, y no por ello menos importante, si no justo lo contrario, el usuario puede acceder a la opción **Nueva Guía**, donde un menú desplegable le ofrecerá la opción de elegir qué tipo quiere crear: Una aplicación exclusivamente para la plataforma Blackberry o una aplicación multiplataforma a través de HTML5.

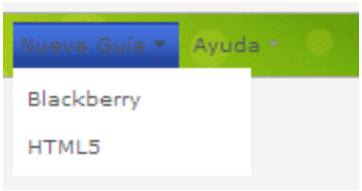


Figura 78: Crear nueva guía

En ambos casos, las pantallas que verá el usuario serán muy similares. Para una mayor comprensión de la funcionalidad, resaltaremos las diferencias llegado el momento. Supongamos que el usuario ha elegido la opción “Blackberry”.

La pantalla que verá a continuación de elegir una opción es la siguiente:

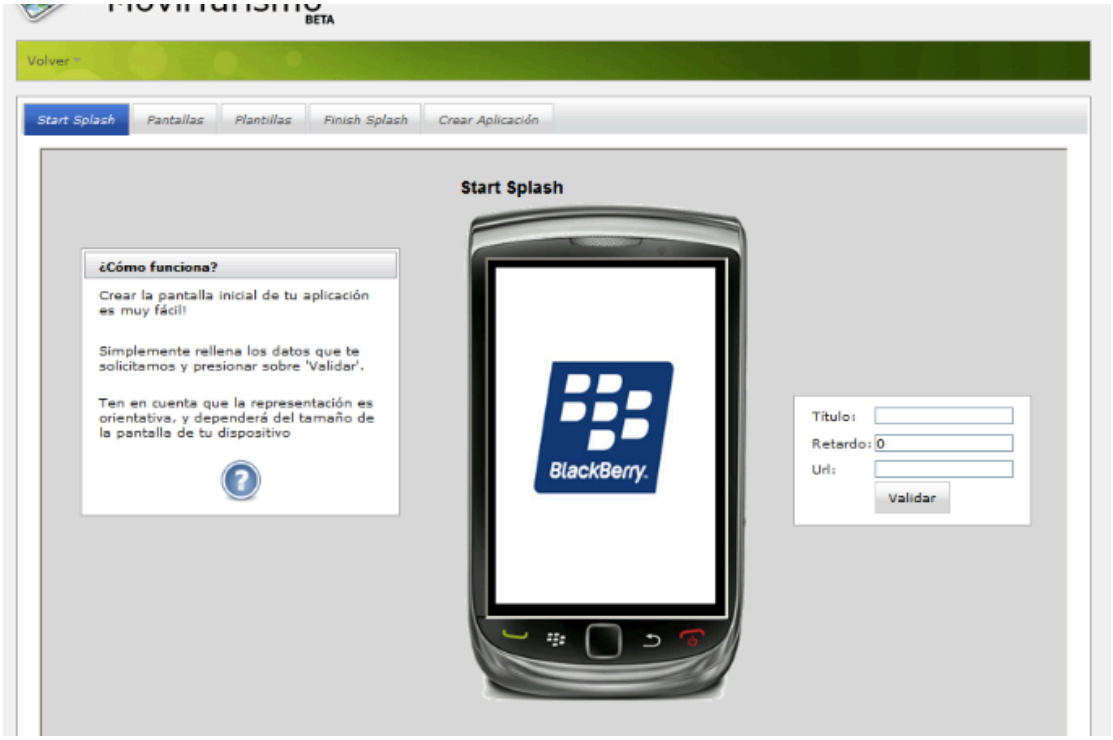


Figura 79: Creación Start Splash

Lo que el usuario tiene en pantalla en estos instantes, recibe el nombre de "Módulo de Generación" y consta de 5 pestañas.

- **Start Splash:** Permite crear la pantalla de presentación de la aplicación
- **Pantallas:** En este apartado se crean todas las pantallas y menús que formarán la aplicación
- **Plantillas:** Permite dar formato a las pantallas creadas. Dependiendo del tipo de aplicación elegida, permite cambiar el fondo o aplicar algunos estilos al texto introducido.
- **Finish Splash:** Permite crear la pantalla final de la aplicación
- **Crear Aplicación:** Desde este apartado, se genera el código final.

En cualquier momento se puede volver al perfil usuario desde la opción *Volver* del menú superior.

7.1. Start Splash

La pestaña **Start Splash** (figura anterior), muestra cómo quedaría visualmente la pantalla inicial o de bienvenida de la aplicación. En la parte izquierda, se le ofrece al usuario una pequeña ayuda a la hora de rellenar los datos que se le solicitan.

El formulario de la parte derecha es la pieza clave para crear la pantalla de inicio. Basta con rellenar los datos que se solicitan y presionar sobre el botón *Validar,* para que se cree la pantalla y se muestre la imagen elegida, a modo de guía visual.

Los datos solicitados son:

- **Título**: título de la pantalla
- **Retardo**: Número de segundos que permanecerá la imagen en pantalla antes de desaparecer.
- **Url**: Url de la imagen a mostrar.

7.2. Pantallas

El siguiente paso, la pestaña **Pantallas**, es la más compleja de la aplicación:

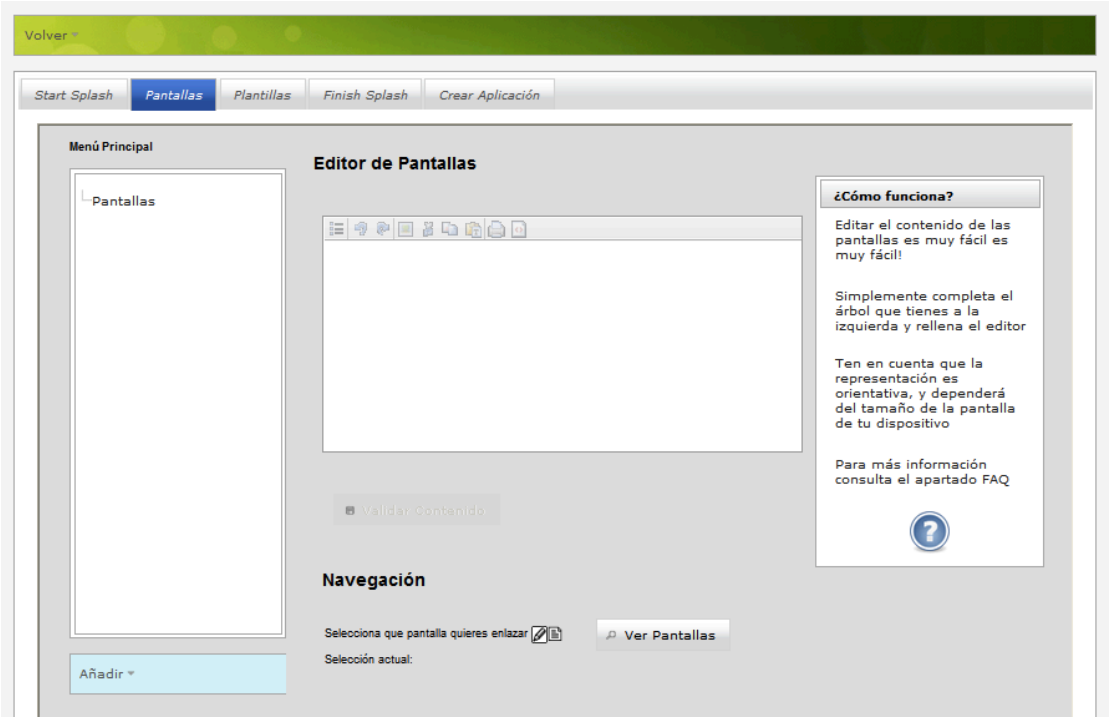


Figura 80: Gestión de Pantallas

En la parte izquierda de la pantalla, se le ofrece al usuario una pequeña ayuda para poder rellenar los valores de la pantalla.

En la parte izquierda, tenemos el componente *Tree*, que será donde el usuario podrá ir creando la jerarquía de pantallas y rellenando su contenido.

En la parte central de la pantalla, tenemos dos apartados: El apartado *"Editor de Pantallas*" donde se encuentra el editor HTML que permitirá al usuario introducir la información de la pantalla y el apartado *"Navegación"*, donde a través del botón "Ver Pantallas" se establece la posibilidad de enlazar una opción de menú con su pantalla correspondiente. Cuando se haya enlazado una pantalla, aparecerá su nombre a continuación de "*Selección actual*:"

Destacar que la única diferencia de esta pantalla de cara a crear la aplicación con HTML5 , es el editor HTML: cuenta con algunos botones más.

El funcionamiento global de la pantalla es el siguiente: El usuario selecciona el nodo "Pantallas" (el nodo padre) del árbol para comenzar a crear la estructura y debe situar el ratón sobre la opción Añadir, que se encuentra en el menú azul situado debajo del componente Tree. Aparecerá un submenú ofreciéndole las opciones *Menú* y *Contenido*.

La opción *Menú* sirve (tal y como su propio nombre indica) para crear entradas de menú. Cuando se crea este tipo de elemento, el editor HTML aparece deshabilitado, puesto que estas opciones no poseen contenido editable. Se representa con . Si la opción menú cuelga directamente del nodo "Pantallas" es decir, pertenece a lo que se considera el menú principal de la aplicación, entonces se representa de la siguiente manera

La opción *Contenido* sirve para crear las pantallas que el usuario podrá modificar y cuyo contenido podrá especificar el usuario. Se representa con

Supongamos que el usuario ha seleccionado la opción Menú: A continuación le aparece un diálogo solicitándole el nombre del menú que va a crear

Figura 81: Insertar nuevo nodo

Pulsando sobre la opción "*Crear*" se creará en el componente *Tree* un nodo de tipo menú principal. Este mecanismo es igual a la hora de crear cualquier nodo.

Puesto que es una entrada de menú, el editor permanecerá deshabilitado, pero sin embargo se habilitará el botón "Ver Pantallas".

Una vez se ha creado un menú y se vuelve a pinchar sobre el mismo nodo padre (en este caso Pantallas), sólo permitirá crear nodos de ese tipo. Lo mismo ocurre con los nodos *Contenido*

Supongamos que ahora el usuario quiere crear un nodo Contenido. Entonces debe seleccionar el que será el nodo padre, situarse sobre la opción "Añadir" y pulsar *Contenido*. De nuevo aparecerá el diálogo solicitando el nombre del nodo y se creará en el árbol colgando del nodo padre que haya elegido el usuario. Para un correcto funcionamiento, no deben repetirse nombres de nodos.

En este caso, el editor HTML se habilitará, permitiendo al usuario la introducción de datos y se deshabilitará la opción "Ver pantallas". Para introducir información, basta con que el usuario escriba lo que desee en el editor (usando los botones disponibles) y pulse sobre el botón "Validar Contenido". En ese momento, se insertará en base de datos el nuevo registro.

De esta manera, el usuario puede ir creando la aplicación como él desee. En la siguiente imagen se muestra un ejemplo de ejecución:

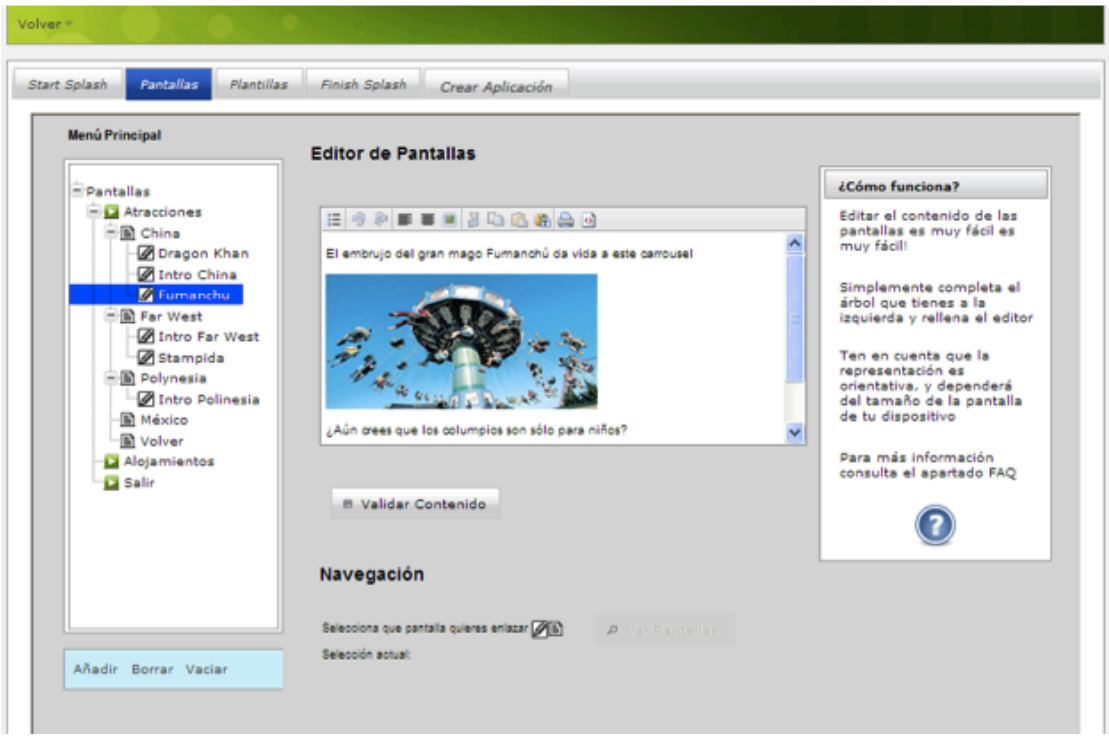


Figura 82: Ejemplo de pantallas

Si fuera necesario borrar un elemento del árbol porque el usuario se ha equivocado al crearlo, posee 2 opciones:

- **Vaciar**: Borrará todo el árbol
- **Borrar**: Permitirá eliminar los nodos uno a uno. La restricción para realizar el borrado es que el nodo seleccionado no tenga nodos hijos creados. Para borrar, basta con seleccionar el nodo en cuestión y situarse sobre la opción Borrar. Si es un nodo apto para el borrado, aparecerá un submenú con la opción "Elemento". Pulsando sobre ella, el nodo desaparecerá del árbol.

Aprovechando la imagen de ejemplo, vamos a suponer que el usuario a creado esa estructura. El siguiente paso es establecer la Navegación. Seleccionando el menú "*China*" y pulsando sobre el botón "*Ver Pantallas*": Aparecerá entonces un diálogo que mostrará las pantallas disponibles para enlazar:

Figura 83: Lista de pantallas a enlazar

Sólo hay que seleccionar la pantalla, a la que queremos acceder desde el menú *China* y pulsar sobre el botón "Aceptar". En este caso suponemos que el usuario selecciona "*Intro China"*.

¿Qué ocurre con las opciones que también cuelgan de China en el árbol? El enlace a las pantallas *Dragon Khan* y *Fumanchú* se realizará desde la siguiente pestaña: la pestaña Plantillas.

7.3. Plantillas

La pestaña **Plantillas** es la encargada de gestionar la apariencia de las pantallas, así como crear los enlaces entre las distintas pantallas de Contenido. La funcionalidad de esta pestaña es muy similar a la anterior:

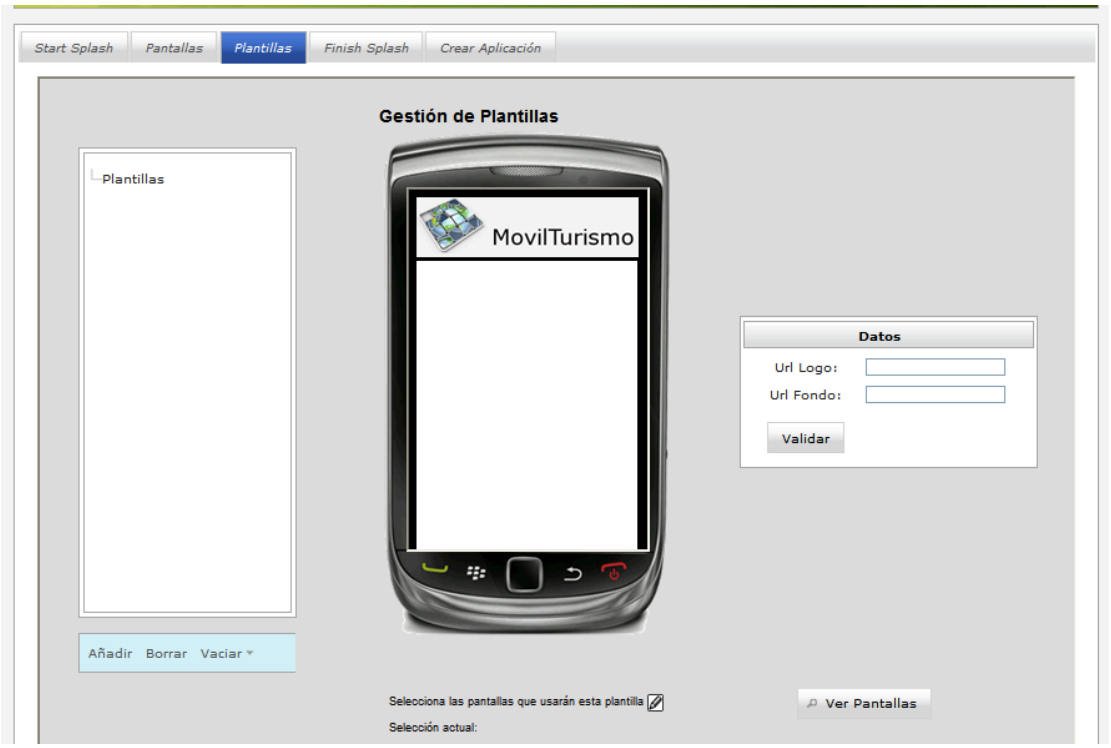


Figura 84: Gestión de Plantillas

En la parte izquierda tenemos el componente *Tree* en el que se irán creando las distintas plantillas. En la parte central, tenemos un "emulador" que nos permitirá ver la apariencia de la plantilla que estamos creando. En, en la parte derecha un formulario, cuyos campos será necesario rellenar para poder crear la plantilla y por último, en la parte inferior, tenemos un botón "Ver Pantallas" que será el encargado de permitirnos realizar los enlaces. Las páginas enlazas a la plantilla se mostrarán junto a la frase "*Selección Actual*"

Tenemos las mismas opciones disponibles en el árbol que anteriormente: Añadir, Borrar y Vaciar. Sin embargo en este caso, no se añaden *Menús* y *Contenido*, si no que se añaden *Plantillas* y *Submenús*.

En la opción *Plantilla* se especifica el color de fondo y el logo que aparecerá en la parte superior de la pantalla. Se representa con

La opción *Submenú* sirve para enlazar las distintas pantallas de contenido. Se representa con

El hecho de enlazar las pantallas aquí, resulta bastante ventajoso pos dos razones: la primera es que es bastante visual para el usuario y de un vistazo puede ver las conexiones entre las pantallas y la segunda, es que permite crear una plantilla única para un gran número de pantallas, por ejemplo, crear una única plantilla para todas las atracciones del área de China.

Del mismo modo que antes, se selecciona el nodo padre y situarse sobre la opción *Añadir*, para seleccionar la opción deseada. Supongamos que se ha elegido la opción "*Plantilla*": aparecerá el mismo diálogo que antes para solicitar el nombre y se creará el nodo.

El siguiente paso es indicar cuál será la imagen de fondo y el logo. Para ello, hay que seleccionar del árbol la plantilla que queremos modificar y rellenar los campos del formulario de la parte derecha de la pantalla. Una vez hecho esto, el usuario debe pulsar sobre el botón "Validar" y aparecerá el emulador cómo quedará la pantalla. Esta acción hace que a la plantilla se le asignen los valores del formulario. Si el usuario se equivoca, puede: o bien borrar el nodo, o bien volver a seleccionar la plantilla del árbol y volver a introducir valores en el formulario.

¿Cómo crear los enlaces que faltan? Seleccionando un nodo padre de tipo plantilla y situando el ratón sobre la opción "Añadir", nos aparecerá la opción "Submenú". En ese momento nos aparecerá el siguiente diálogo:

Figura 85: Enlace a pantalla desde plantilla

En él, podremos elegir la pantalla que queremos enlazar. Recordemos que teníamos pendiente enlazar la pantalla *Dragon Khan* y *Fumanchú*. Elegimos una de estas dos opciones y automáticamente se creará el enlace.

Aparecerá bajo el nodo de la plantilla elegida, un nodo de tipo submenú cuyo nombre será el nombre de la pantalla enlace concatenado con *_sub* al final. En el siguiente ejemplo puede verse la creación de los enlaces

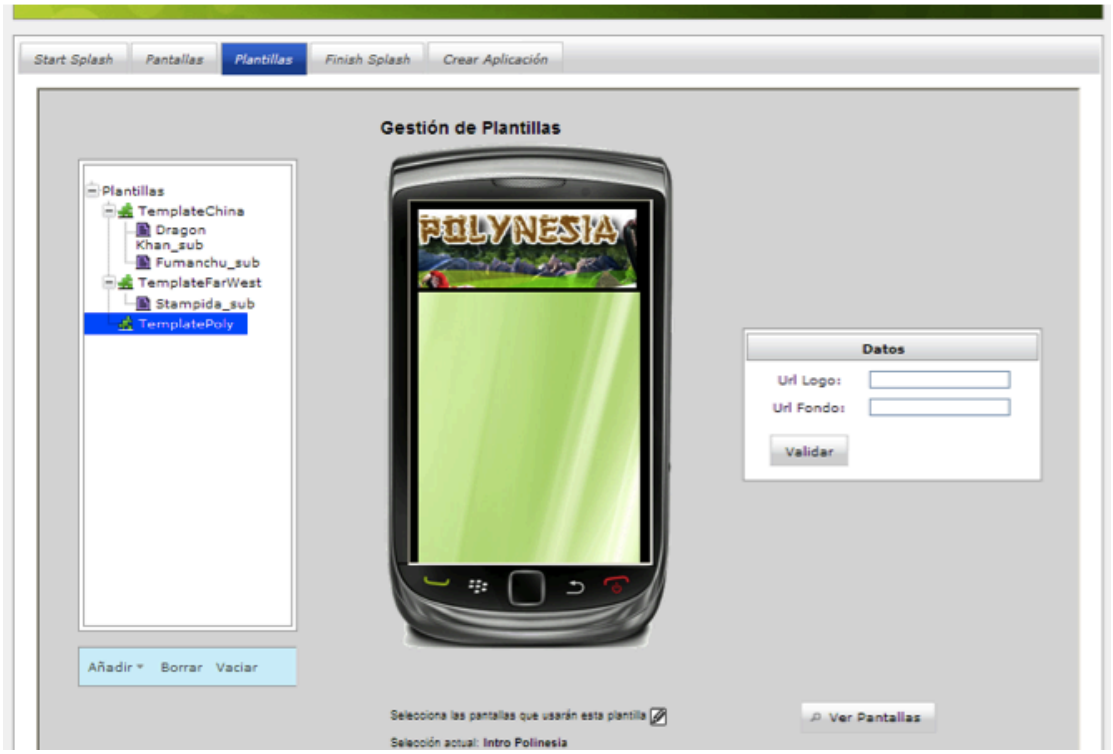


Figura 86: Ejemplo pestaña Plantillas (I)

Desde la plantilla "*TemplateChina*", que usarán las pantallas *Intro China, Dragon Khan y Fumanchú*, se podrá acceder a las opciones *Dragon Khan* y *Fumanchú* de manera permanente.

Existen diferencias significativas entre los dos tipos de plataforma en esta pantalla. La apariencia si estamos creando una aplicación con HTML5 es la siguiente:

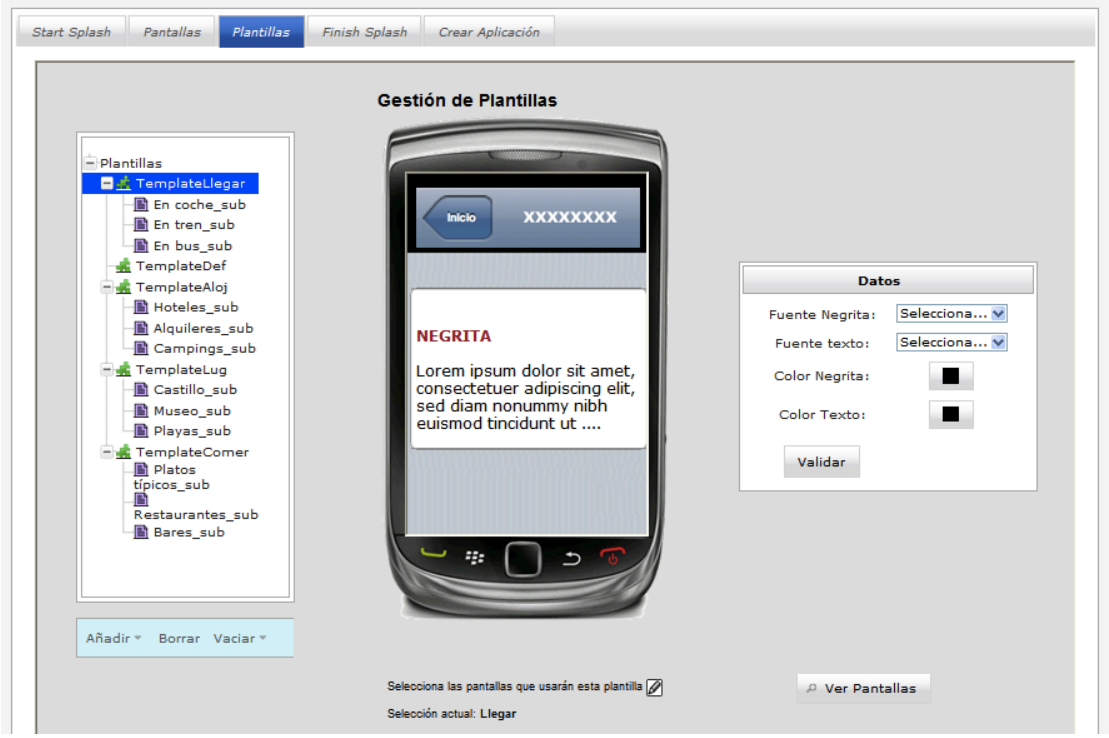


Figura 87: Ejemplo pestaña Plantillas (II)

Se ha sustituido el formulario de la pantalla anterior, por otro que permita indicar el tipo de letra que aparecerá en la aplicación y el color. No se permite modificar el fondo ni el logo para darle un toque más actual.

7.4. Finish Splash

La pestaña **Finish Splash** es como la pestaña Start Splash. En esta pestaña se muestra cómo quedaría visualmente la pantalla final o de despedida de la aplicación. En la parte izquierda, se le ofrece al usuario una pequeña ayuda a la hora de rellenar los datos que se le solicitan.

El formulario de la parte derecha es la pieza clave para crear la pantalla final. Basta con rellenar los datos que se solicitan y presionar sobre el botón *Validar* para que se cree la pantalla y se muestre la imagen elegida a modo de guía visual.

Los datos solicitados son:

- **Título**: título de la pantalla
- **Retardo**: Número de segundos que permanecerá la imagen en pantalla antes de desaparecer y cerrar la aplicación
- **Url**: Url de la imagen a mostrar.

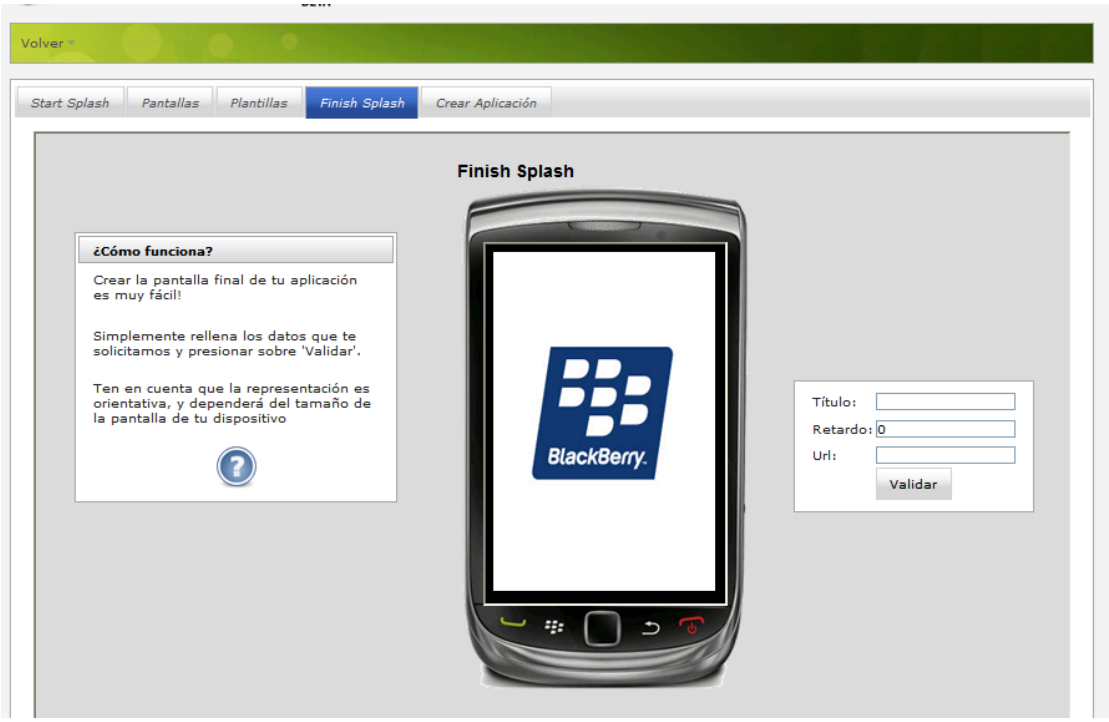


Figura 88: Creación Finish Splash

7.5. Crear Aplicación

Por último, la pestaña **Crear Aplicación** es donde se genera el código final de la aplicación. En la parte izquierda, aparece un panel *Informe* en el que se mostrarán los resultados de la generación del código. Si todo ha ido correctamente, aparecerá la frase "¡Generación correcta!", en caso contrario aparecerá una lista de los errores encontrados.

Estos errores pueden ser por ejemplo, que no haya imagen definida en el start splash, o que alguna pantalla no tenga contenido, etc…

Cuando se pulsa sobre el botón "*Empezar*", comienza a generarse el código. Nos aparece un diálogo informativo indicando que el proceso ha comenzado:

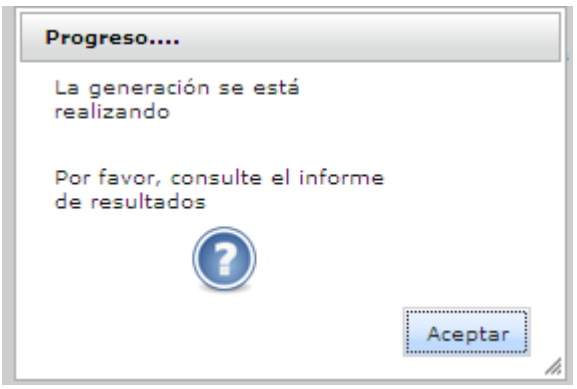


Figura 89: Diálogo de progreso

Cuando finalice y si todo ha ido correctamente, se habrá generado el código. Dependiendo de la plataforma para la que se esté generando, se accederá al código de una manera u otra.

En caso de ser Blackberry, la aplicación nos llevará a la carpeta *tmp* del ordenador, donde podremos coger el código, compilarlo en la suite correspondiente y subirlo a la tienda de aplicaciones de Blackberry.

En caso contrario, si estamos generando para HTML5, la aplicación colgará directamente el código resultante en el servidor, y nos permitirá ejecutarla inmediatamente con un navegador, haciéndola disponible para el usuario desde el momento en que el código termina de generarse.

Figura 90: Pestaña Crear Aplicación

8. Conclusiones

En este apartado del documento, comentaremos las conclusiones que hemos deducido de la realización de la presente tesis.

El uso cada vez mayor de los Smartphones permite gestionar la información que consideramos más esencial para realizar nuestras actividades diarias, con la ventaja de tenerla siempre a mano en cualquier momento. No es de extrañar que grandes y pequeñas empresas intenten hacerse hueco en este mercado.

Sin embargo, el hecho de que un usuario pueda crear su propia aplicación sin la intervención de terceros y sin conocimiento en programación, es una característica en auge. Existe un gran número de soluciones Web que permiten este cometido. Claro está que por el momento, la complejidad de las aplicaciones generadas no es demasiado elevada, pero irá incrementándose con el tiempo.

En este marco, destacamos la arquitectura MDA, como una arquitectura tremendamente potente, cuyo uso presenta gran cantidad de ventajas y el *desarrollo End-Use*r, orientado a que sea el usuario final de la aplicación el encargado de crearla.

En un intento por aportar un granito de arena más a este escenario, hemos realizado un estudio de las soluciones que existen en la red que permiten al usuario generar su propia aplicación. Dicho estudio nos ha permitido concebir una aplicación que intenta aprovechar las ventajas de las soluciones existentes: MovilTurismo, presentada como una plataforma de

desarrollo en la que cualquier usuario registrado puede generar código para cualquier plataforma existente.

Dicha plataforma permitirá al usuario generar aplicaciones móviles compatibles con Blackberry (nativo) y HTML5, de tal manera que una aplicación generada podrá usarse tanto en móviles con el sistema operativo Blackberry, como en cualquier dispositivo que posea un navegador compatible con HTML5, acercándose de esta manera a la multiplataforma.

MovilTurismo ha sido programado con tecnologías actuales y en constante evolución para ofrecer al usuario una aplicación dinámica, atractiva y sobretodo usabilidad. Esto implica un mayor esfuerzo durante el proceso de codificación, ya que se han tratado con tecnologías desconocidas y ha sido necesario un periodo de aprendizaje previo para poder crear MovilTurismo.

La usabilidad es una característica muy relevante, que hemos ido buscando durante todo el desarrollo de la plataforma. En el trabajo previo hemos visto que se puede generar código de una manera similar… pero no es nada intuitivo para el usuario. Es más, es complicado. Es por eso, que se ha creado esta plataforma Web: para que la aplicación sea usable de cara a usuarios finales.

En conclusión, hemos querido ofrecer conjuntamente las ventajas de MDA y del desarrollo *End-User* para aportar una solución más al abanico de posibilidades que la red nos ofrece para este tipo de desarrollos.

9. Trabajo Futuro

Como en todo el software desarrollado, siempre existen mejoras que pueden aportarse: en este caso, se considera que una de las mejoras principales que debe incorporar la aplicación es el reconocimiento de más componentes por parte del editor HTML. En concreto, elementos multimedia como sonido o video, así como, añadir las funcionalidades típicas con las que cuentan las aplicaciones móviles actuales: mapas, contactos, etc.

MovilTurismo es capaz de generar aplicaciones en Blackberry nativo y HTML5. Sin embargo, sería deseable ampliar la variedad e incluir más plataformas destino que mejorarían las opciones de desarrollo.

Además, en la actualidad el funcionamiento está pensado para ordenadores con sistema operativo Windows puesto que se realiza el volcado del código en la carpeta tmp del ordenador. Una versión futura, debería ser capaz de volcar la aplicación en servidor y permitir descargarla en formato zip o rar, de tal manera que pudiese ejecutarse en cualquier SO.

Del mismo modo, debería ser compatible con cualquier tipo de navegador y que su interfaz gráfica, se mantuviera estable y sin ningún cambio.

Dada la gran relevancia de las redes sociales y el impacto que tienen hoy en día, sería muy interesante hacer la plataforma web más social: los usuarios podrían compartir conocimiento, guías y ayudarse a solventar las dudas, así como probar la aplicación, de tal manera que podría existir un contínuo *feedback* entre usuarios finales y desarrolladores, en pos de realizar los ajustes necesarios a la plataforma y estudiar los resultados obtenidos durante su uso.

Estas mejoras, harán que la próxima versión de MovilTurismo, genere aplicaciones que se encuentren dentro de la funcionalidad esperada hoy en día en los smartphones

10. Bibliografía

[1] http://zintegra.blogspot.com/2007/07/generacin-automtica-de-cdigo.html

[2] http://www.pros.upv.es/index.php/es/lineas/69-lineaddm

[3] *Desarrollo de Procesos de Negocio Móviles Adaptados a la Obtrusividad* - Rocío Carolina Martínez Arenas (Diciembre 2009)

[4] *Introduciendo a los Usuarios Finales en el Desarrollo de Sistemas Pervasivos. Una aproximación MDD* – María Francisca Pérez Pérez (Diciembre 2009)

[5] http://appadvice.com/appnn/2011/06/breaking-400000-ios-applications-app-store

[6] http://www.xatakandroid.com/mercado/android-market-sobrepasa-las-250000-aplicaciones

[7] http://www.movilevento.com/

[8] http://www.mobimento.com/caponate.php

[9] http://appinventor.googlelabs.com/about/

[10] http://www.buzztouch.com/pages/

[11] http://www.tekora.com/en/

[12] http://www.phonegap.com/

[13] http://www.buzzle.com/editorials/7-18-2004-56792.asp

[14] http://pewinternet.org/

[15] http://blog.shoutem.com/2011/04/21/infographic-the-state-of-mobile-app-world/

[16] *Aplicación para BlackBerry: Guía de Portaventura* – Beatriz Marchante Banegas (Julio 2010)

[17] Capítulo 3: JavaServer Faces: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/viveros_s_ca/capitulo3.pdf

[18] Tutorial de JavaServer Faces: http://www.sicuma.uma.es/sicuma/Formacion/documentacion/JSF.pdf

[19] http://www.primefaces.org/

[20] Guía de usuario de Primefaces: primefaces_users_guide_140210.pdf

[21] http://grupos.emagister.com/ficheros/dspflashview?idFichero=93779

[22] http://es.wikipedia.org/wiki/Tomcat

[23] http://www.esepestudio.com/articulo/desarrollo-web/bases-de-datos-mysql/Que-es-MySQL.htm

[24] http://www.openarchitectureware.org/pub/documentation/4.3.1/html/contents/workflow_reference.html

[25] http://commons.apache.org/io/

[26] PFC Gestión de una dieta alimenticia vía Web – Rosa María Trapero Alcaide / Beatriz Marchante Banegas (Junio 2007)

[27] http://blog.entornao.com/web20/2008/01/29/web-como-plataforma/

[28] *Eclipse Modeling Framewok (EMF) Sistemas Distribuidos* – Diego Sevilla Ruiz (Octubre 2010) - http://ditec.um.es/ssdd/eclipse-emf.pdf

[29] *Desarrollo por el Usuario Final* - José Antonio Macías Iglesias http://www.aipo.es/articulos/5/1361.pdf

[30] http://www.moskitt.org/cas/que-es-moskitt/

ANEXO 1 – ECORE

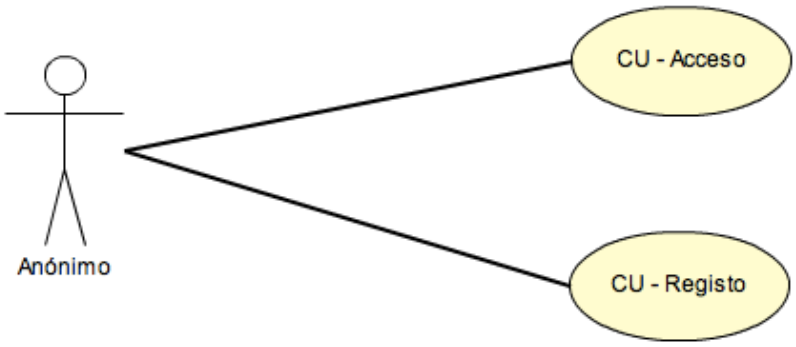
ANEXO 2 – MODELO DE CASOS DE USO

Vamos a ver los casos de uso más significativos de MovilTurismo así como los actores que existen en el sistema.

Los actores identificados en la aplicación son los que se detallan a continuación:

- **Anónimo**: podrá registrarse en la aplicación o acceder, si posee una clave de acceso.
- **Registrado:** podrá realizar todas las acciones disponibles de la aplicación

Vemos a continuación los casos de uso que forman parte del usuario *Anónimo*



CU – Registro

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario se registre en la aplicación

Precondiciones: -

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Anónimo	Acceder a la página Web de MovilTurismo	Se accederá a la pantalla principal
2	Anónimo	Rellenar los campos que se solicitan	Los campos quedarán rellenos
3	Anónimo	Pulsar sobre el botón “Registro”	Se efectuará el registro en el

ORDEN	ACTOR	ACCIÓN	REACCIÓN
			sistema y el usuario accederá a la pantalla de su perfil (CU – Ver Perfil del usuario registrado)

Tabla 12: CU- Registro

Restricciones: -

CU – Acceso

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario se valide en la aplicación

Precondiciones: -

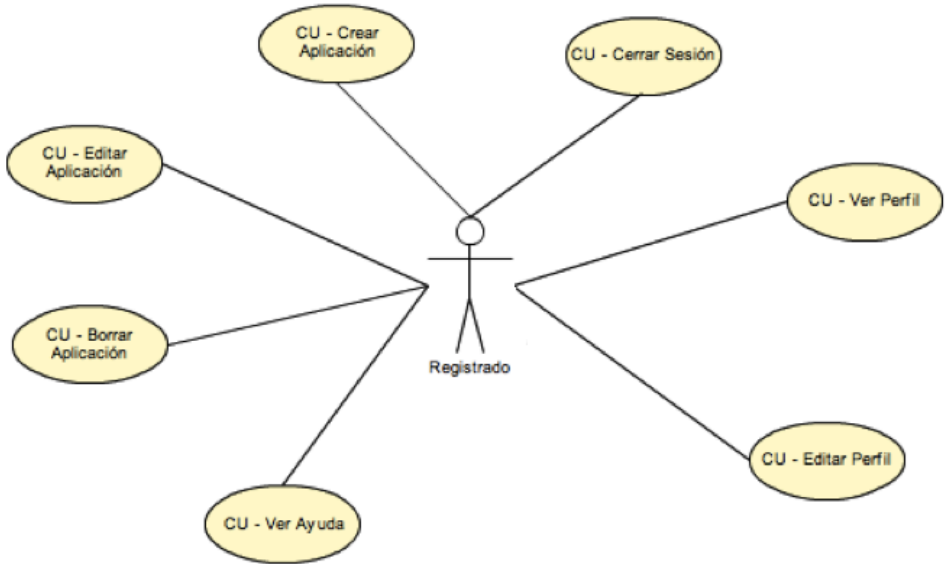
Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Anónimo	Acceder a la página Web de MovilTurismo	Se accederá a la pantalla principal
2	Anónimo	Rellenar los campos “Usuario” y “Password”	Los campos quedarán rellenos
3	Anónimo	Pulsar sobre el botón “Login”	Se realizará la comprobación en base de datos y se redireccionará a la página del perfil. En caso contrario aparecerá información del error (usuario y password incorrectos)

Tabla 13: CU - Acceso

Restricciones: El usuario debe de estar dado de alta en el sistema.

De la misma manera, vemos a continuación los casos de uso que forman parte del usuario *Registrado.* Debido al gran número de casos de uso, separaremos los diagramas en dos grupos uno pertenece a los casos de uso accesibles desde el perfil del usuario y el otro pertenece a los casos de uso pertenecientes a la propia generación de la aplicación.



CU – Ver Perfil

Descripción: En este caso de uso, se describen las acciones a realizar para visualizar el perfil de usuario

Precondiciones: Haberse validado correctamente en la aplicación

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	-	La aplicación muestras los datos del perfil del usuario validado y las aplicaciones que ha creado hasta el momento.

Tabla 14: CU - Ver Perfl

Restricciones: -

CU – Editar Perfil

Descripción: En este caso de uso, se describen las acciones a realizar para editar el perfil del usuario

Precondiciones: Pasar por el CU – Ver Perfil

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Seleccionar la opción “Mi Perfil” situada en la barra de menús de la parte superior	El sistema mostrará las acciones disponibles
2	Registrado	Presionar sobre la opción “Modificar”	Se accederá a la pantalla de modificación. Se cargarán los datos del usuario en pantalla en campos editables
3	Registrado	Modificar los campos en pantalla	Los campos quedarán modificados
4	Registrado	Presionar sobre el botón “Cambiar Imagen”	Aparecerá un diálogo para introducir la url de la imagen
5	Registrado	En el diálogo: Pulsar sobre el botón “Aceptar”	Se cambiará la imagen y se cerrará el diálogo
6	Registrado	En el diálogo: Pulsar sobre el botón “Cancelar”	No se cambiará la imagen y se cerrará el diálogo
7	Registrado	Presionar sobre el botón “Modificar”	Se realizará la modificación en base de datos

Tabla 15: CU - Editar Perfil

Restricciones: -

CU – Cerrar Sesión

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario cierre su sesión

Precondiciones: Pasar por el CU – Ver Perfil

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Seleccionar la opción “Mi Perfil” situada en la barra de menús de la parte superior	El sistema mostrará las acciones disponibles
2	Registrado	Presionar sobre la opción “Cerrar Sesión”	Se cerrará la sesión del usuario y se redireccionará a la pantalla principal de la aplicación

Tabla 16: CU - Cerrar Sesión

Restricciones: -

CU – Crear Aplicación

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario cree una nueva aplicación

Precondiciones: Pasar por el CU – Ver Perfil

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Seleccionar la opción "Nueva Guía" situada en la barra de menús de la parte superior	El sistema mostrará las acciones disponibles
2	Registrado	Presionar sobre la opción correspondiente al tipo de aplicación que se va a crear (BlackBerry / HTML5)	Se accederá al módulo de generación del tipo de la aplicación seleccionada. En concreto la aplicación se posicionará en el CU –Start Splash.

Tabla 17: CU - Crear Aplicación

Restricciones: -

CU – Editar Aplicación

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario pueda continuar con la edición de una aplicación ya creada.

Precondiciones: Pasar por el CU – Ver Perfil

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Seleccionar sobre una fila de la tabla de aplicaciones creadas	Se marcará la fila
2	Registrado	Pulsar sobre el botón 	Se cargarán los datos de base de datos de la aplicación seleccionada y se accederá al caso de uso CU – Start Splash.

Tabla 18: CU - Editar Aplicación

Restricciones: Tener una fila de la tabla seleccionada

CU – Borrar Aplicación

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario pueda borrar una aplicación creada.

Precondiciones: Pasar por el CU – Ver Perfil

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Seleccionar sobre una fila de la tabla de aplicaciones creadas	Se marcará la fila
2	Registrado	Pulsar sobre el botón 	Se borrará la aplicación

Tabla 19: CU - Borrar Aplicación

Restricciones: Tener una fila de la tabla seleccionada

CU – Ver Ayuda

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario consulte la ayuda

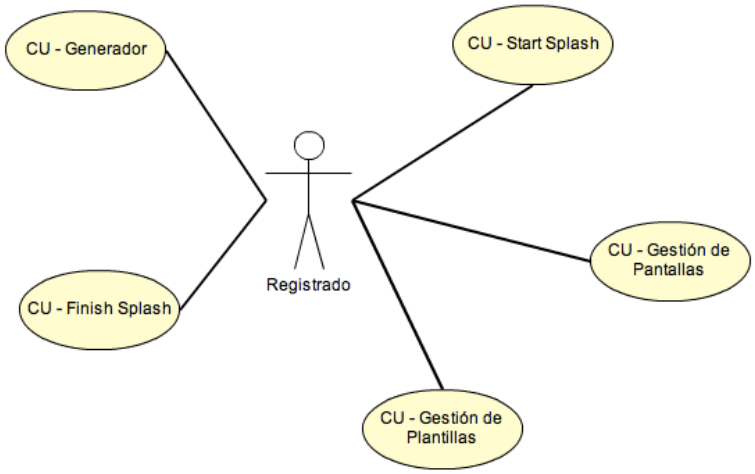
Precondiciones: Pasar por el CU – Ver Perfil

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Seleccionar la opción "Ayuda" situada en la barra de menús de la parte superior	El sistema mostrará las acciones disponibles
2	Registrado	Presionar sobre la opción "FAQ"	El sistema accederá a la página de ayuda para el usuario

Tabla 20: CU - Ver Ayuda

Restricciones: -



CU – Start Splash

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario cree la pantalla inicial de la aplicación

Precondiciones: Pasar por el CU – Crear Aplicación o CU – Editar Aplicación y situarse en la pestaña "Start Splash"

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Rellenar los campos "Título", "Retardo" y "url"	Los valores quedarán rellenos
2	Registrado	Presionar sobre la opción "Validar"	Se mostrará la vista previa de la url de la imagen y se insertarán / actualizarán los datos en la base de datos

Tabla 21: CU - Start Splash

Restricciones: -

CU – Gestión de Plantillas

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario pueda crear las plantillas de la aplicación

Precondiciones: Pasar por el CU – Crear Aplicación o CU – Editar Aplicación y situarse en la pestaña "Plantillas"

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Seleccionar un nodo	El nodo quedará seleccionado
2	Registrado	Situarse sobre la opción “Añadir”	Se mostrarán las opciones disponibles.
3	Registrado	Pulsar sobre la opción “Plantilla”	Aparecerá un diálogo solicitando el nombre de la plantilla
4	Registrado	En el diálogo, pulsar sobre el botón “Crear”	Se creará el nuevo nodo
5	Registrado	Desde paso 2: Pulsar sobre la opción “Submenú”	Aparecerá un diálogo con el árbol de pantallas que permite seleccionar que pantalla vamos a enlazar.
6	Registrado	En el diálogo: Pulsar sobre el botón “Aceptar”	Se realiza el enlace
7	Registrado	En el diálogo: Pulsar sobre el botón “Cancelar”	Se cancela la acción
8	Registrado	Desde paso 1: Situarse sobre la opción “Borrar”	Si es un nodo sin hijos, permitirá seleccionar la opción “Elemento” y aparecerá un diálogo de confirmación
9	Registrado	En el diálogo: Pulsar sobre el botón “Aceptar”	El nodo se borrará
10	Registrado	En el diálogo: Pulsar sobre el botón “Cancelar”	Se cancela la acción
11	Registrado	Situarse sobre la opción “Vaciar / Todo”	Aparecerá un diálogo de confirmación
12	Registrado	En el diálogo: Pulsar sobre el botón “Aceptar”	El árbol se vaciará
13	Registrado	En el diálogo: Pulsar sobre el botón “Cancelar”	Se cancela la acción
14	Registrado	Seleccionar un nodo de tipo Plantilla	El nodo queda seleccionado, se habilitará el botón “Ver Pantallas” y aparecerán información de las pantallas seleccionadas en “Selección actual” (si la hubiese)
15	Registrado	Rellenar las opciones que tipo de letra y color que ofrece la pantalla y pulsar sobre el botón “Validar”	Aparecerá una vista previa de la selección y se almacenará la plantilla de base de dato
16	Registrado	Pulsar sobre el botón “Ver pantallas”	Aparecerá una lista de selección múltiple que permitirá elegir qué pantallas van a usar la plantilla
17	Registrado	En la lista:	Se realiza la asignación y las

ORDEN	ACTOR	ACCIÓN	REACCIÓN
		Pulsar sobre el botón “Aceptar”	pantallas aparecerán como “Selección actual”
18	Registrado	En la lista: Pulsar sobre el botón “Cancelar”	Se cancela la acción

Tabla 22: CU - Gestión de Plantillas

Restricciones:

- Para añadir, siempre debe permanecer el nodo padre seleccionado.
- Sólo se permite borrar la última hoja de la rama, es decir, aquella que no tiene hijos.
- No pueden existir nombres de ramas repetidos
- Sólo se permitirá crear la opción “Plantilla” si se selecciona el nodo padre
- Sólo se permitirá crear la opción “Submenú” si el nodo seleccionado es de tipo “Plantilla”

CU – Gestión de Pantallas

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario pueda crear las pantallas de la aplicación

Precondiciones: Pasar por el CU – Crear Aplicación o CU – Editar Aplicación y situarse en la pestaña “Pantallas”

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Seleccionar un nodo	El nodo quedará seleccionado
2	Registrado	Situarse sobre la opción “Añadir”	Se mostrarán las opciones disponibles.
3	Registrado	Pulsar sobre la opción “Menú”	Aparecerá un diálogo solicitando el nombre del menú
4	Registrado	En el diálogo, pulsar sobre el botón “Crear”	Se creará el nuevo nodo
5	Registrado	Desde paso 2: Pulsar sobre la opción “Contenido”	Aparecerá un diálogo solicitando el nombre de la pantalla de contenido
6	Registrado	En el diálogo, pulsar sobre el botón “Crear”	Se creará el nuevo nodo
7	Registrado	Desde paso 1:	Si es un nodo sin hijos, permitirá

ORDEN	ACTOR	ACCIÓN	REACCIÓN
		Situarse sobre la opción “Borrar”	seleccionar la opción “Elemento” y aparecerá un diálogo de confirmación
8	Registrado	En el diálogo: Pulsar sobre el botón “Aceptar”	El nodo se borrará
9	Registrado	En el diálogo: Pulsar sobre el botón “Cancelar”	Se cancela la acción
10	Registrado	Situarse sobre la opción “Vaciar / Todo”	Aparecerá un diálogo de confirmación
11	Registrado	En el diálogo: Pulsar sobre el botón “Aceptar”	El árbol se vaciará
12	Registrado	En el diálogo: Pulsar sobre el botón “Cancelar”	Se cancela la acción
13	Registrado	(Con el árbol creado) Seleccionar un nodo Menú	Se habilitará el botón “Ver Pantallas” y mostrará la “Selección Actual” (si existe)
14	Registrado	Pulsar sobre el botón “Ver pantallas”	Aparecerá un diálogo que nos permitirá seleccionar la pantalla hacia la que se navegará
15	Registrado	En el diálogo: Pulsar sobre el botón “Aceptar”	Se realiza el enlace y aparecerá el nombre de la pantalla a continuación de “Selección Actual”
16	Registrado	En el diálogo: Pulsar sobre el botón “Cancelar”	Se cancela la acción
17	Registrado	(Con el árbol creado) Seleccionar un nodo Contenido	Se habilitará el editor para introducir datos y el botón “validar Contenido”
18	Registrado	Rellenar la información en el editor y pulsar sobre el botón “Validar Contenido”	Se almacenará en base de datos la pantalla

Tabla 23: CU - Gestión de Pantallas

Restricciones:

- -Para añadir, siempre debe permanecer el nodo padre seleccionado.
- -Sólo se permite borrar la última hoja de la rama, es decir, aquella que no tiene hijos.
- -No pueden existir nombres de ramas repetidos
- -Sólo se permitirá crear la opción “Menú” si para el nodo seleccionado no se han creado elementos “Contenido”

- -Sólo se permitirá crear la opción “Contenido” si para el nodo seleccionado no se han creado elementos “Menú”.

CU – Finish Splash

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario cree la pantalla final de la aplicación

Precondiciones: Pasar por el CU – Crear Aplicación o CU – Editar Aplicación y situarse en la pestaña “Finish Splash”

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Rellenar los campos “Título”, “Retardo” y “url”	Los valores quedarán rellenos
2	Registrado	Presionar sobre la opción “Validar”	Se mostrará la vista previa de la url de la imagen y se insertarán / actualizarán los datos en la base de datos

Tabla 24: CU - Finish Splash

Restricciones: -

CU – Generador

Descripción: En este caso de uso, se describen las acciones a realizar para que el usuario genere el código de la aplicación.

Precondiciones: Pasar por el CU – Crear Aplicación o CU – Editar Aplicación y situarse en la pestaña “Crear Aplicación”

Escenario:

ORDEN	ACTOR	ACCIÓN	REACCIÓN
1	Registrado	Presionar sobre la opción “Empezar”	El sistema mostrará un diálogo indicando que el proceso de generación a comenzado
2	Registrado	En el diálogo, presionar sobre el botón “Aceptar”	Cuando acabe el proceso, se mostrará el resultado del mismo en el cuadro de texto “Informe”
3	Registrado	Presionar sobre el enlace /botón	Se accederá al código/ aplicación

ORDEN	ACTOR	ACCIÓN	REACCIÓN
		(según plataforma)	generada.

Tabla 25: CU - Generador

Restricciones: -

ANEXO 3 – SCRIPT DE BASE DE DATOS

Estructura de tabla para la tabla `wm_aplicacion`

```
CREATE TABLE IF NOT EXISTS `wm_aplicacion` (
  `aplicacion_id` int(10) NOT NULL AUTO_INCREMENT,
  `fecha_crea` date NOT NULL,
  `fecha_actu` date NOT NULL,
  `aplicacion_nom` varchar(150) NOT NULL,
  `usuario_id` int(10) NOT NULL,
  `plataforma` varchar(10) NOT NULL,
  `generada` varchar(200) NOT NULL,
  PRIMARY KEY (`aplicacion_id`),
  KEY `usuario_id` (`usuario_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Estructura de tabla para la tabla `wm_informe`

```
CREATE TABLE IF NOT EXISTS `wm_informe` (
  `id_informe` int(11) NOT NULL AUTO_INCREMENT,
  `id_aplicacion` int(11) NOT NULL,
  `textoProblema` varchar(500) NOT NULL,
  `rutaicono` varchar(150) NOT NULL,
```

```
  PRIMARY KEY (`id_informe`),
  KEY `id_aplicacion` (`id_aplicacion`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Estructura de tabla para la tabla `wm_menu`

```
CREATE TABLE IF NOT EXISTS `wm_menu` (
  `id_menu` int(11) NOT NULL AUTO_INCREMENT,
  `nombre_menu` varchar(150) NOT NULL,
  `id_creacion` int(11) DEFAULT NULL,
  `id_aplicacion` int(11) NOT NULL,
  `nombre_padre` varchar(150) NOT NULL,
  `contador` int(11) NOT NULL,
  PRIMARY KEY (`id_menu`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Estructura de tabla para la tabla `wm_screen`

```
CREATE TABLE IF NOT EXISTS `wm_screen` (
  `id_screen` int(11) NOT NULL AUTO_INCREMENT,
  `id_aplicacion` int(11) NOT NULL,
  `titulo` varchar(100) NOT NULL,
  `id_padre` int(11) DEFAULT NULL,
  `plantilladef` varchar(1) NOT NULL,
  `plantillanom` varchar(150) DEFAULT NULL,
  `nombre` varchar(100) NOT NULL,
  `target` varchar(100) DEFAULT NULL,
  `fechains` datetime NOT NULL,
  `tipo` varchar(3) NOT NULL,
  `contenidohtml` varchar(15000) DEFAULT NULL,
```

```
  `nombre_padre` varchar(150) NOT NULL,
  `contador` int(11) NOT NULL,
  `url_logo` varchar(500) DEFAULT NULL,
  `url_fondo` varchar(500) DEFAULT NULL,
  `activo` varchar(5) NOT NULL,
  PRIMARY KEY (`id_screen`),
  KEY `id_padre` (`id_padre`),
  KEY `id_aplicacion` (`id_aplicacion`),
  KEY `titulo` (`titulo`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Estructura de tabla para la tabla `wm_splash`

```
CREATE TABLE IF NOT EXISTS `wm_splash` (
  `start_id` int(10) NOT NULL AUTO_INCREMENT,
  `aplicacion_id` int(10) NOT NULL,
  `titulo` varchar(150) DEFAULT NULL,
  `segundos` int(10) NOT NULL,
  `source` varchar(500) NOT NULL,
  `tipo` varchar(20) NOT NULL,
  `tipo_splash` varchar(3) NOT NULL,
  PRIMARY KEY (`start_id`),
  KEY `aplicacion_id` (`aplicacion_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Estructura de tabla para la tabla `wm_submenu`

```
CREATE TABLE IF NOT EXISTS `wm_submenu` (
  `id_submenu` int(11) NOT NULL AUTO_INCREMENT,
  `id_aplicacion` int(11) NOT NULL,
```

```
  `id_pantalla_ref` int(11) NOT NULL,
  `id_template_ref` int(11) NOT NULL,
  `fechains` datetime NOT NULL,
  `nombre` varchar(50) NOT NULL,
  PRIMARY KEY (`id_submenu`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Estructura de tabla para la tabla `wm_template`

```
CREATE TABLE IF NOT EXISTS `wm_template` (
  `id_template` int(11) NOT NULL AUTO_INCREMENT,
  `id_aplicacion` int(11) NOT NULL,
  `titulo` varchar(50) NOT NULL,
  `fechains` datetime NOT NULL,
  `contador` int(11) NOT NULL,
  `nombre_padre` varchar(50) NOT NULL,
  `url_logo` varchar(500) NOT NULL,
  `url_fondo` varchar(500) NOT NULL,
  `activo` varchar(1) NOT NULL,
  `style` varchar(200) NOT NULL,
  `stylebold` varchar(200) NOT NULL,
  PRIMARY KEY (`id_template`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Estructura de tabla para la tabla `wm_usuarios`

```
CREATE TABLE IF NOT EXISTS `wm_usuarios` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(100) NOT NULL,
```

```
  `apellido1` varchar(100) NOT NULL,
  `apellido2` varchar(100) DEFAULT NULL,
  `mail` varchar(100) NOT NULL,
  `login` varchar(50) NOT NULL,
  `password` varchar(50) NOT NULL,
  `fechareg` date NOT NULL,
  `url_img` varchar(500) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

Filtros para la tabla `wm_aplicacion`

```
ALTER TABLE `wm_aplicacion`
ADD CONSTRAINT `wm_aplicacion_ibfk_1` FOREIGN KEY (`usuario_id`)
REFERENCES `wm_usuarios` (`id`);
```

ANEXO 4 – PLANTILLAS DE CONVERSIÓN

Plantilla para BlackBerry

```
«IMPORT MobileMetamodel»

«DEFINE principal FOR Application»
        «FILE "Application.java"»
        import net.rim.device.api.ui.UiApplication;

        class Application extends UiApplication {
   Application() {
     pushScreen(new StartSplashScreen(this,new MenuScreen(this)));
   }

    public static void main(String[] args){
     Application myApp = new Application();
     myApp.enterEventDispatcher();
   }
}
        «ENDFILE»
        «EXPAND claseJava FOREACH Screens»
        «EXPAND claseTemp FOREACH templates»
«ENDDEFINE»
«DEFINE claseJava FOR Screen»
        «FILE name+".java"»
                public class «name» {
```

```
                    private int id = «id»;
                    private String name = "«name»";
                    private String title = "«tittle»";
                }
        «ENDFILE»
«ENDDEFINE»
«DEFINE claseJava FOR StartScreen»
«FILE name+".java"»
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import java.util.*;

        public class «name» extends MainScreen {
                    private int id = «id»;
                    private String name = "«name»";
                    private String title = "«tittle»";
                    private int transition = «transitionSeconds»;
                    private Bitmap imagen =
Bitmap.getBitmapResource("«Images.get(0).source»");
                    private MainScreen next;
                    private UiApplication application;
                    private Timer timer = new Timer();

                    public «name» (UiApplication ui, MainScreen next) {
                        super(Field.USE_ALL_HEIGHT |
Field.USE_ALL_WIDTH );
                        this.application = ui;
```

```
                                        this.next = next;

                                        BitmapField bf = new BitmapField (imagen);
                                        bf.setSpace (Graphics.getScreenWidth()/2 -
imagen.getWidth()/2,
                                        Graphics.getScreenHeight()/2 - imagen.getHeight()/2
);
                                        this.add (bf);

                                        SplashScreenListener listener = new
SplashScreenListener(this);
                                        this.addKeyListener(listener);
                                         timer.schedule(new CountDown(), transition);

                                }

                                public void dismiss() {
                                application.popScreen(this);
                                application.pushScreen(next);
                                        }

                                private class CountDown extends TimerTask {
                                 public void run() {
                            DismissThread dThread = new DismissThread();
                            application.invokeLater(dThread);
                           }

                }

                private class DismissThread implements Runnable {
```

```
 public void run() {
dismiss();
}
}
protected boolean navigationClick(int status, int time) {
dismiss();
return true;
}

protected boolean navigationUnclick(int status, int time) {
return false;
}

protected boolean navigationMovement(int dx, int dy, int status, int time) {
 return false;
 }

 public static class SplashScreenListener implements
KeyListener {
private «name» screen;
public boolean keyChar(char key, int status, int time) {
  boolean retval = false;
switch (key) {
 case Characters.CONTROL_MENU:
case Characters.ESCAPE:
screen.dismiss();
retval = true;
break;
```

```
    }
    return retval;
  }
  public boolean keyDown(int keycode, int time) {
    return false;
  }
  public boolean keyRepeat(int keycode, int time) {
    return false;
  }
  public boolean keyStatus(int keycode, int time) {
    return false;
  }
  public boolean keyUp(int keycode, int time) {
    return false;
  }
  public SplashScreenListener(«name» splash) {
    screen = splash;
  }
 }


            }
    «ENDFILE»
    «ENDDEFINE»

«DEFINE claseJava FOR MenuScreen»
«FILE name+".java"»
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
```

```
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import java.util.Vector;
import java.util.*;

public class «name» extends MainScreen {
                    private UiApplication application;
                    private int id = «id»;
                    private String name = "«name»";
                    private String title = "«tittle»";
                    private Vector mainMenuItems;
          private MenuListField menu;
          private String img;

public «name»(UiApplication ui) {
        super();
        this.mainMenuItems = new Vector();
        this.setTitle("Menú");
         this.application = ui;

«FOREACH MenuElements AS me»
        this.mainMenuItems.addElement("«me.value»");
        «IF me.Icon != null»
        this.img = "«me.Icon.source»";
        «ENDIF»
        «ENDFOREACH»

        String[] items = new String[this.mainMenuItems.size()];
```

```
        for(int i = 0; i < this.mainMenuItems.size(); i++){
            items[i] =  this.mainMenuItems.elementAt(i).toString();
        }
menu = new MenuListField(items,  Bitmap.getBitmapResource(img));
        this.add(menu);
    }

    protected boolean navigationClick(int status, int time) {

        // Determine which menu item was clicked.
        int selected = menu.getSelectedIndex();
        switch (selected) {
         «FOREACH MenuElements AS me»
            case «me.id»:
                application.pushScreen(new «me.TargetScreen.name»(application));
            break;
         «ENDFOREACH»
        }
             return true;
    }

public class MenuListField extends ObjectListField {
    private String[] items;
    private Bitmap icon;

            public  MenuListField (String[] menuListItems, Bitmap resource)
            {           super();
                        this.set(menuListItems);
```

```
                items = menuListItems;
                icon = resource;
            }

    // We are going to take care of drawing the item.
    public void drawListRow(ListField listField, Graphics graphics,
            int index, int y, int width) {

        if (null != icon) {

            graphics.drawBitmap(1,y , icon.getWidth(),
                icon.getHeight(),icon,0,0);
            graphics.drawText(items[index],
                icon.getWidth() + 2, y, DrawStyle.ELLIPSIS, width);
} else {
            graphics.drawText("- " + items[index], 0,
                y, DrawStyle.ELLIPSIS, width);
        }
    }
}
                                        }
```

«ENDFILE»

«ENDDEFINE»

«DEFINE claseJava FOR **FinishScreen**»

«FILE name+".java"»

```
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import java.util.*;

public class «name» extends MainScreen {
        public static final int waitMiliseconds= 1500;
        private static final Bitmap _bitmap =
Bitmap.getBitmapResource("«this.Images.get(0).source»");
  private UiApplication application;
  private Timer timer = new Timer();

  public EndSplashScreen(UiApplication ui) {
   super(Field.USE_ALL_HEIGHT | Field.USE_ALL_WIDTH );//|
Field.FIELD_HCENTER| Field.FIELD_VCENTER);
    this.application = ui;

   /*las siguientes 4 lineas colocan la imagen en el medio de la pantalla */
    BitmapField bf = new BitmapField (_bitmap );
    bf.setSpace (Graphics.getScreenWidth()/2 - _bitmap.getWidth()/2,
    Graphics.getScreenHeight()/2 - _bitmap.getHeight()/2 );

   this.add (bf);

    SplashScreenListener listener = new SplashScreenListener(this);
    this.addKeyListener(listener);
    timer.schedule(new CountDown(), waitMiliseconds);
```

```
}

public void dismiss() {
        System.exit(0);
}
private class CountDown extends TimerTask {
  public void run() {
    DismissThread dThread = new DismissThread();
    application.invokeLater(dThread);
  }
}

private class DismissThread implements Runnable {
  public void run() {
    dismiss();
  }
}

protected boolean navigationClick(int status, int time) {
  dismiss();
  return true;
}

protected boolean navigationUnclick(int status, int time) {
  return false;
}

protected boolean navigationMovement(int dx, int dy, int status, int time) {
```

```
    return false;
  }

  public static class SplashScreenListener implements
    KeyListener {
    private EndSplashScreen screen;
    public boolean keyChar(char key, int status, int time) {
      //intercept the ESC and MENU key - exit the splash screen
      boolean retval = false;
      switch (key) {

        case Characters.CONTROL_MENU:
        case Characters.ESCAPE:
        screen.dismiss();
        retval = true;
        break;
      }
      return retval;
    }

    public boolean keyDown(int keycode, int time) {
      return false;
    }

    public boolean keyRepeat(int keycode, int time) {
      return false;
    }
```

```
    public boolean keyStatus(int keycode, int time) {
      return false;
    }

    public boolean keyUp(int keycode, int time) {
      return false;
    }

    public SplashScreenListener(EndSplashScreen splash) {
      screen = splash;
    }

    }
        }

        «ENDFILE»
        «ENDDEFINE»

        «DEFINE claseJava FOR RegularScreen»

        «FILE name+".java"»
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.system.Bitmap;
import java.util.Vector;

public class «name»
```

```
«IF this.screen_templ.size == 0»
  extends MainScreen  {
  «ELSE»
  extends «this.screen_templ.get(0).name» {
  «ENDIF»

protected UiApplication application;
boolean haySub = false;

 public «name»(UiApplication ui) {
                        super(ui, "«tittle»");
                        this.application = ui;
                        «EXPAND divisionTemplate FOREACH Divisions»
}

public final class ListCallback implements ListFieldCallback {
            private Vector listElements = new Vector();

            public void drawListRow(
            ListField list, Graphics g, int index, int y, int w) {
            String text = (String)listElements.elementAt(index);

            g.drawText(" *  "+text, 15, y, 0, w);
            }
            public Object get(ListField list, int index) {
            return listElements.elementAt(index);
            }
            public int indexOfList(ListField list, String p, int s) {
```

```
            return listElements.indexOf(p, s);
            }
            public int getPreferredWidth(ListField list) {
            return Graphics.getScreenWidth();
            }
            public void insert(String toInsert, int index) {
            listElements.addElement(toInsert);
            }
            public void erase() {
            listElements.removeAllElements();
            }
            }
}
            «ENDFILE»
            «ENDDEFINE»

«DEFINE divisionTemplate FOR Division»
«EXPAND p FOREACH  DivisionElements»
«ENDDEFINE»

«DEFINE p FOR Pharagraf»
«EXPAND t FOREACH PharagrafElements»
«ENDDEFINE»

«DEFINE t FOR Text»
RichTextField «name» = new RichTextField("«value»");
                        this.AddFieldToContent(«name»);
«ENDDEFINE»
```

```
«DEFINE p FOR Image»

Bitmap bitmap_image_1«name» = Bitmap.getBitmapResource("«source»");

                BitmapField bitmapField_image_1«name» = new
BitmapField(bitmap_image_1«name», BitmapField.FIELD_HCENTER);

                this.AddFieldToContent(bitmapField_image_1«name»);

«ENDDEFINE»


«DEFINE p FOR Lista»

int cont«name» = 0;

ListField listfield1«name» = new ListField(2,NON_FOCUSABLE| READONLY);


                ListCallback myCallback«name» = new ListCallback();

                listfield1«name».setCallback(myCallback«name»);

«EXPAND s FOREACH subElements»


this.AddFieldToContent(listfield1«name»);

«ENDDEFINE»


«DEFINE s FOR ListElement»

myCallback.insert("«value»", cont«name»);

cont«name»++;

«ENDDEFINE»

«DEFINE p FOR Element»

«ENDDEFINE»

«DEFINE claseTemp FOR Template»

        «FILE name + ".java"»

import net.rim.device.api.ui.*;

import net.rim.device.api.ui.component.*;
```

```
import net.rim.device.api.ui.container.*;

import net.rim.device.api.system.*;

import java.util.Vector;


		public class «name» extends MainScreen {


	private static final Bitmap LogoImage=
Bitmap.getBitmapResource("«Head.Logo.source»");

	private static String ApplicationNAME = "«Head.Tittle.value»";

	private static final Bitmap BackGroundImage =
Bitmap.getBitmapResource("«BackGround.source»");


private UiApplication application;

	private VerticalFieldManager MainHolder;

	private HorizontalFieldManager HeadingHolder;

	private VerticalFieldManager ContentHolder;

	private HorizontalFieldManager HeadingFooter;

	RichTextField colorField;


	Menu menu;

	Vector vector = new Vector();


	public «name»(UiApplication ui, String name) {

		super(MainScreen.NO_VERTICAL_SCROLL);


		application = ui;

		this.menu = new Menu();

		if(name != null && name != ""){

		this.ApplicationNAME = name;
```

```
                }


                MainHolder = new
VerticalFieldManager(VerticalFieldManager.USE_ALL_HEIGHT|
VerticalFieldManager.USE_ALL_WIDTH)


    {
        //Override the paint method to draw the background image.
         public void paintBackground(Graphics graphics) {
            int startX = 0;
            int startY = 0;


            // Determine where to center image
            if (this.getVisibleWidth() > BackGroundImage.getWidth()) {
               startX = (this.getVisibleWidth() - BackGroundImage.getWidth()) / 2;
            }
            if (this.getVisibleHeight() > BackGroundImage.getHeight()) {
               startY = (this.getVisibleHeight() - BackGroundImage.getHeight()) / 2;
            }


            //Draw the background image and then call paint.
            graphics.drawBitmap(startX, startY, BackGroundImage.getWidth(),
BackGroundImage.getHeight(), BackGroundImage, 0, 0);


            super.paintBackground(graphics);
        }
    };
    this.add(MainHolder);
HeadingHolder = new
HorizontalFieldManager(HorizontalFieldManager.NO_HORIZONTAL_SCROLL
```

```
        | HorizontalFieldManager.NO_VERTICAL_SCROLL |
HorizontalFieldManager.USE_ALL_WIDTH){


        protected void paintBackground(Graphics graphics)
        {
                graphics.setBackgroundColor(000000);
                graphics.clear();
                super.paint(graphics);
        }
    };


    //Application logo
    if (LogoImage!=null)
    {
     MainHolder.add(new BitmapField(LogoImage));
    }


    //Application Title
    if (ApplicationNAME !=null)
    {
        FontFamily ff;
                        try {
                                ff = FontFamily.forName("System");




        final Font fnt = ff.getFont(Font.BOLD, 8, Ui.UNITS_pt);
colorField = new RichTextField(ApplicationNAME) {
                // Override the paint method to set Font and Color.
```

```
                public void paint(Graphics graphics) {
                  setFont(fnt);
                  //setBackground(imageBackground);
                  graphics.setColor(Color.WHITE);
                  super.paint(graphics);
                }
        };      } catch (ClassNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
    }
HeadingHolder.add(colorField);
     MainHolder.add(HeadingHolder);

    ContentHolder = new
VerticalFieldManager(VerticalFieldManager.VERTICAL_SCROLL
        | VerticalFieldManager.VERTICAL_SCROLLBAR |
VerticalFieldManager.USE_ALL_WIDTH)

    {
    };

     ContentHolder.setPadding(0,0,10,10);
    MainHolder.add(ContentHolder);

//Add-Footer----------------------------------------------------
 //A HorizontalFieldManager to hold the footer;
   HeadingFooter = new
HorizontalFieldManager(HorizontalFieldManager.NO_HORIZONTAL_SCROLL
```

```
        | HorizontalFieldManager.NO_VERTICAL_SCROLL |
HorizontalFieldManager.USE_ALL_WIDTH);

MainHolder.add(HeadingFooter);

«EXPAND p FOR Footer»

        }

        public void AddFieldToContent(Field field)

        {

                ContentHolder.add(field);

        }

        public void SetHeaderTittle()

        {}


        public void SetHeaderLogo()

        {}


        public void AddFieldToHeader()

        {

        }

        public void AddFieldToFooter()

        {

        }

        public void BackScreen()

        {

                application.popScreen(this);

                //application.pushScreen(new MenuScreen(application) );

        }


        public void CloseScreen()

        {
```

```
        application.popScreen(this);
        application.pushScreen(new EndSplashScreen(application));

        //TODO application.pushScreen(new SplashScreenEnd(application) );
}

private MenuItem backItem = new MenuItem("Volver", 100, 10) {
        public void run() {
                BackScreen();
        }
};

private MenuItem closeItem2 = new MenuItem("Salir", 100, 10) {
        public void run() {
        //onClose();
                CloseScreen();
        }
};

protected void makeMenu(Menu menu, int instance) {
for(int i = 0; i < this.vector.size() ; i++){
menu.add((MenuItem)this.vector.elementAt(i));

}

menu.add(MenuItem.separator(100));
menu.add(backItem);
menu.add(closeItem2);
```

```
        menu.add(MenuItem.separator(100));
        }
«EXPAND se FOR Footer»
        }
«ENDFILE»
«ENDDEFINE»

«DEFINE p FOR Footer»
«EXPAND p FOR Submenu»
«ENDDEFINE»

«DEFINE p FOR Submenu»
«EXPAND s FOREACH  Submenus»
«ENDDEFINE»

«DEFINE s FOR SubMenuElement»
        this.vector.addElement(new MenuItem("«name»", 100, 10) {
                        public void run() {
                                SubItem«this.Screens.name»Function();
                        }
         });
«ENDDEFINE»

«DEFINE se FOR Footer»
«EXPAND se FOR Submenu»
«ENDDEFINE»

«DEFINE se FOR Submenu»
```

«EXPAND se FOREACH Submenus»

«ENDDEFINE»

«DEFINE se FOR **SubMenuElement**»

```
        private void SubItem«this.Screens.name»Function()
        {
                application.popScreen(this);
                application.pushScreen(new «this.Screens.name»(application));
        }
```

«ENDDEFINE»

Plantilla para HTML5

```
«IMPORT MobileMetamodel»

«DEFINE principal FOR Application»
        «FILE "acceso.html"»
        «ENDFILE»
        «EXPAND claseJava FOREACH Screens»
        «EXPAND claseTemp FOREACH templates»


«ENDDEFINE»


«DEFINE claseJava FOR Screen»
        «FILE name+".html"»


public class «name» {
                                private int id = «id»;
                                private String name = "«name»";
                                private String title = "«tittle»";
                }
        «ENDFILE»
«ENDDEFINE»


«DEFINE claseJava FOR StartScreen»
«FILE "index.html"»
<!DOCTYPE html>
<html lang="es">
```

```
<head>
        <META HTTP-EQUIV= "Content-Type"CONTENT="text/html;charset= ISO-8859-1">
   <link rel="shortcut icon" href="../imagenes/deniaIcon.png" />
   <title>«name»</title>
   <meta name="Propietario" content="Laboratorios del Centro de Investigación ProS - www.pros.upv.es">
   <meta name="Version" content="1.0">
   <link href="../../css/style.css" rel="stylesheet" media="screen" type="text/css" />
   <meta content="minimum-scale=1.0, width=device-width, maximum-scale=0.6667, user-scalable=no" name="viewport" />
</head>
<body>
        <section id="topbar">
                <section id="title">«name»</section>
        <section id="leftnav"></section>
      <section id="rightnav"><a href="index2.html">Entrar</a></section>
        </section>

        <section id="content">
                <article class="pageitem">
           <article class="textbox">
                <p class="centeredParagraph">
                <img class="mainImg" src="«this.Images.get(0).source»">
            </p>
         </article>
                </article>
        </section>
</body>
```

```
</html>

        «ENDFILE»

«ENDDEFINE»


«DEFINE claseJava FOR MenuScreen»

«FILE name+".html"»

<!DOCTYPE html>

<html lang="es">

<head>

        <META HTTP-EQUIV= "Content-Type"CONTENT="text/html;charset= ISO-8859-1">

   <link rel="shortcut icon" href="../imagenes/deniaIcon.png" />

   <title>«name»</title>

   <meta name="Propietario" content="Laboratorios del Centro de Investigación ProS - www.pros.upv.es">

   <meta name="Version" content="1.0">

   <link href="../../css/style.css" rel="stylesheet" media="screen" type="text/css" />

   <meta content="minimum-scale=1.0, width=device-width, maximum-scale=0.6667, user-scalable=no" name="viewport" />

</head>

<body>

        <section id="topbar">

        <section id="title">Guía Turística</section>

      <section id="leftnav"><a href="index.html"><img alt="home" src="../../images/home.png" /></a></section>

      <section id="rightnav"></section>

   </section>

   <section id="content">

        <article class="pageitem">
```

```
      «FOREACH MenuElements AS me»
    <article class="menu">
          «IF me.value == "Salir"»
          <a href="finish.html">
          «ELSE»
           <a href="«me.valueTarget».html">
          «ENDIF»
                  <span class="name">«me.value»</span>
              <span class="arrow"></span>
           </a>
          </article>
           «ENDFOREACH»
</article>



</section>
</body>
</html>
          «ENDFILE»
«ENDDEFINE»

«DEFINE claseJava FOR FinishScreen»

«FILE "finish.html"»
<!DOCTYPE html>
<html lang="es">
<head>
```

```
        <META HTTP-EQUIV= "Content-Type"CONTENT="text/html;charset= ISO-8859-1">
   <link rel="shortcut icon" href="../imagenes/deniaIcon.png" />
   <title>«name»</title>
   <meta name="Propietario" content="Laboratorios del Centro de Investigación ProS - www.pros.upv.es">
   <meta name="Version" content="1.0">
   <link href="../../css/style.css" rel="stylesheet" media="screen" type="text/css" />
   <meta content="minimum-scale=1.0, width=device-width, maximum-scale=0.6667, user-scalable=no" name="viewport" />
</head>
<script type="text/javascript">

setTimeOut(cerrar(), «transitionSeconds»00)

function cerrar(){
  window.close();
}
</script>
<body onload="cerrar()">
        <section id="topbar">
                <section id="title">¡Adios!</section>
        <section id="leftnav"></section>
        </section>
        <section id="content">
                <article class="pageitem">
          <article class="textbox">
                <p class="centeredParagraph">
                <img class="mainImg" src="«this.Images.get(0).source»">
```

```
            </p>
        </article>
                </article>
        </section>
</body>
</html>

        «ENDFILE»
«ENDDEFINE»

«DEFINE claseJava FOR RegularScreen»
«FILE name+".html"»
<!DOCTYPE html>
<html lang="es">
<head>
        <META HTTP-EQUIV= "Content-Type"CONTENT="text/html;charset= ISO-8859-1">
   <link rel="shortcut icon" href="../imagenes/deniaIcon.png" />
   <title>Guía Turística de Denía</title>
   <meta name="Propietario" content="Laboratorios del Centro de Investigación ProS - www.pros.upv.es">
   <meta name="Version" content="1.0">
   <link href="../../css/style.css" rel="stylesheet" media="screen" type="text/css" />
        <meta content="minimum-scale=1.0, width=device-width, maximum-scale=0.6667, user-scalable=no" name="viewport" />
</head>
<body>
«IF this.screen_templ.Footer.Submenu.Submenus.size == 0»
<section id="topbar">
```

```
        <section id="title">Guía Turística</section>
    <section id="leftnav"><a href='javascript:history.go(-1)'>Volver</a></section>

    <section id="rightnav"></section>
        </section>

 «ELSE»
 <section id="topbar">
        <section id="title">Guía Turística</section>
    <section id="leftnav"><a href="index2.html">Inicio</a></section>
    <section id="rightnav"></section>
        </section>

        <nav id="tributton">
                <section class="links">
                «FOREACH this.screen_templ.Footer.Submenu.Submenus AS sub»<a
href="«sub.name».html">«sub.value»</a>«ENDFOREACH»
    </section>
        </nav>
«ENDIF»

<section id="content">
        <article class="pageitem">
        <article class="textbox">
                «EXPAND divisionTemplate FOREACH Divisions»

                </article>
                </article>
```

```
        </section>
</body>
</html>

        «ENDFILE»
«ENDDEFINE»

«DEFINE divisionTemplate FOR Division»
«EXPAND p FOREACH  DivisionElements»
«ENDDEFINE»

«DEFINE p FOR Image»
<p class="centeredParagraph" id = "«name»">
                <img class="mainImg" src="«source»">
 </p>
 <BR>
«ENDDEFINE»

«DEFINE p FOR Pharagraf»
«EXPAND p FOREACH PharagrafElements»
«ENDDEFINE»

«DEFINE p FOR Text»
«IF bold == "S"»
 <p id = "«name»" style = "«stylebold»"><B>«value»</B></p>
 <BR>
  «ELSE»
```

```
 «IF ref == "N"»
 «IF value != " "»
 <p id = "«name»" style = "«style»">«value»</p>
 <BR>
 «ENDIF»
 «ELSE»
   «IF mostrar == "S"»
 </article>
 </article>
        <article class="pageitem">
        «ENDIF»
                <article class="menu">
 <a href="«ref»"><span class="name">«value»</span></a></article>
 «ENDIF»



«ENDIF»
«ENDDEFINE»

«DEFINE p FOR Lista»
<p>
<UL>
«EXPAND s FOREACH subElements»
</UL>
</p>
«ENDDEFINE»
```

«DEFINE s FOR **ListElement**»

<LI>«value»</LI>

«ENDDEFINE»

«DEFINE p FOR **Element**»

«ENDDEFINE»

«DEFINE claseTemp FOR **Template**»

«ENDDEFINE»