UNIVERSIDAD INTERNACIONAL DE LA RIOJA

TRABAJO FINAL
MASTER EN DIRECCIÓN E INGENIERÍA DE SITIOS WEB

Generación de guías turísticas en HTML5 mediante un desarrollo dirigido por modelos

Autor: **Francisco Núñez Sánchez**
Director: **Begoña Cristina Pelayo García-Bustelo**

Madrid 2012

Resumen

El objetivo principal ha sido estudiar los fundamentos del desarrollo dirigido por modelos (MDA), para luego implementar una aplicación. Para ello, se han investigados los principios y fundamentos del desarrollo dirigido por modelos y se ha enfocado al uso de XSLT como lenguaje de transformación.

Se ha implementado una aplicación de creación de guías turísticas que genera código HTML5. Para el desarrollo se ha utilizado el lenguaje de programación C#, XML, XML Schema y XSLT.

Palabras Clave: *Desarrollo dirigido por modelos (MDA), XSLT, guías turísticas, HTML5.*

Abstract

The main purpose was to study the fundamentals of model-driven development (MDA) and then, implement an application. Therefore, we investigated the principles and basis of model-driven development and has focused on the use of XSLT transformation language.

We have implemented an authoring application that generates code guidebooks HTML5. For development, we used the programming language C#, XML, XML Schema and XSLT.

Keywords*: Model-driven development (MDA), XSLT, guidebooks, HTML5.*

Agradecimientos

A mi compañera

Especialmente quiero dedicar este trabajo a los profesores del Máster, los cuales me han aportado nuevos conocimientos que de seguro me ofrecerán nuevas oportunidades en esta compleja ingeniería.

A mi familia, por soportarme siempre.

A Flora por su carácter dócil y amable.

Para investigar la verdad es preciso dudar, en cuanto sea posible, de todas las cosas, una vez en la vida.

René Descartes.

Tabla de Contenidos

Tabla de Figuras

Tabla de Códigos

Índice de Tablas

PARTE I

Introducción a la Investigación

Capítulo 1
Introducción

En esta primera sección se va a introducir de manera resumida los aspectos principales de este proyecto. En primer lugar, se muestran las motivaciones que nos han llevado a realizar este trabajo; en segundo lugar, se explican cuáles son los objetivos que se han establecido sobre él; por último, se comentan las secciones en las que está dividida esta memoria, junto a una descripción breve de su contenido.

1.1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL TRABAJO

Dado el carácter del presente trabajo, la finalidad primordial por la cual se ha elaborado es la presentación del mismo como Trabajo de Fin de Máster.

Otra de las motivaciones que me ha impulsado a elaborar este proyecto es el descubrir y probar aplicaciones orientadas al programador totalmente nuevas para mí e intentar encontrar una manera óptima de crear aplicaciones dadas las exigencias actuales de la pequeña y mediana empresa.

Las exigencias del usuario van aumentando con el tiempo debido a las nuevas tecnologías. Actualmente en el mercado existen multitud de lenguajes de programación y una gran variedad de utilidades *y frameworks* que ayudan al programador a que sus tareas sean más fáciles tras su previo aprendizaje. Para intentar enfrentarnos a este problema tecnológico surge el desarrollo de software dirigido por modelos, que pretende usar los modelos, usados por ahora únicamente como documentación, como las entidades principales del desarrollo de software.

1.2. HIPÓTESIS Y OBJETIVOS

Los objetivos de este proyecto son los siguientes:

- Estudiar cuáles son los principios y fundamentos de MDA.
- Investigar sobre la creación de plantillas XSLT para la generación de código HTML5.
- Por último, integración de la/s plantilla/s en una aplicación sencilla para usuario sin conocimientos de programación.
- Creación de guía turística haciendo uso de las plantillas creadas.

1.3. METODOLOGÍA SEGUIDA DURANTE LA INVESTIGACIÓN

Para el desarrollo del proyecto me he basado en la metodología ágil AUP (*Agile Unified Process*) [AMBL09].

El proceso unificado ágil (AUP) es una versión simplificada de RUP desarrollada por Scott Ambler. Describe un enfoque simple, fácil de entender, del desarrollo de software de aplicación de negocios usando técnicas y conceptos ágiles. AUP aplica técnicas ágiles incluyendo desarrollo orientado a pruebas, modelado ágil, gestión de cambios ágil y refactorización de bases de datos para mejorar la productividad. La naturaleza en serie de AUP se presenta en cuatro fases:

- Inicio: el objetivo es identificar el alcance inicial del proyecto, una arquitectura potencial para el sistema y obtener fondos y aceptación por parte de las personas involucradas en el negocio.
- Elaboración: el objetivo es probar la arquitectura del sistema.
- Construcción: el objetivo es construir software operativo de forma incremental que cumpla con las necesidades de prioridad más altas de las personas involucradas en el negocio.
- Transición: el objetivo es validar y desplegar el sistema en el entorno de producción.

AUP tiene siete disciplinas:

1. Modelado. Entender el negocio de la organización, tratar el dominio del problema e identificar una solución viable para tratar el dominio del problema.
2. Implementación. Transformar el modelo en código ejecutable y realizar un nivel básico de pruebas, en particular pruebas unitarias.
3. Pruebas. Realizar una evaluación objetiva para asegurar calidad. Esto incluye encontrar defectos, validar que el sistema funciona como fue diseñado y verificar que se cumplen los requisitos.
4. Despliegue. Planificar el despliegue del sistema y ejecutar el plan para poner el sistema a disposición de los usuarios finales.
5. Gestión de configuración. Gestión de acceso a los artefactos del proyecto. Esto no sólo incluye el seguimiento de las versiones de los artefactos sino también controlar y gestionar los cambios en ellos.
6. Gestión de proyecto. Dirección de las actividades que tienen lugar dentro del proyecto. Esto incluye gestionar riesgos, dirigir y coordinar a las personas y sistemas fuera del alcance del proyecto para asegurar que se entrega a tiempo y dentro del presupuesto.
7. Entorno. Soporte del resto del esfuerzo asegurando que el proceso, la orientación (estándares y guías) y las herramientas (software, hardware…) adecuadas están disponibles para el equipo cuando son necesarias.

Para realizar el análisis de los requisitos, además de los diagramas E-R y casos de uso he utilizado UWE (*UML-based Web Engineering*) [LUDW10].

UWE es un método orientado a objetos basado en UML, que es usado para la especificación de aplicaciones web.

El enfoque UWE proporciona un proceso de desarrollo dirigido por modelos y es una herramienta de apoyo para el diseño de aplicaciones.

El método consta de seis modelos:

1. Modelo de casos de uso para capturar los requisitos del sistema.
2. Modelo conceptual para el contenido (modelo del dominio).
3. Modelo de usuario: modelo de navegación que incluye modelos estáticos y dinámicos.
4. Modelo de estructura de presentación, modelo de flujo de presentación.
5. Modelo abstracto de interfaz de usuario y modelo de ciclo de vida del objeto.
6. Modelo de adaptación.

1.4. ORGANIZACIÓN DEL TRABAJO DE FIN DE MÁSTER

A continuación se describe brevemente el contenido de los capítulos y apartados de esta memoria:

- Capítulo 1: Introducción

  Objetivos principales, metodología, motivaciones y la organización de la presente memoria.

- Capítulo 2: Estado del arte

En este apartado se explica en qué se basa y qué es el desarrollo dirigido por modelos y la arquitectura de modelos. Al hilo de este tipo de desarrollos se procede a comentar los lenguajes de transformación, más concretamente el que hace uso este proyecto, XSLT y otros lenguajes relacionados.

- Capítulo 3. Desarrollo del proyecto

  En este capítulo se expone el desarrollo del proyecto y la aplicación, su implementación y pruebas.

- Capítulo 4: Conclusiones y Líneas Futuras

  Aquí se comenta lo aprendido durante el desarrollo del proyecto, así como las conclusiones al finalizar su implementación. En este apartado también se detallan algunos aspectos ampliables a la aplicación desarrollada, como pueden ser añadir funcionalidad y puntos en los que se puede mejorar.

- Capítulo 5: Referencias

  Aquí se muestra tanto la bibliografía utilizada como páginas de Internet y artículos relacionados con los temas utilizados en el desarrollo del proyecto, ya sean temas informáticos o especializados en la materia de la aplicación.

PARTE II

Estado del Arte

Capítulo 2
Estado del arte

En este apartado se hace un repaso a las tecnologías relacionadas con el estudio. En particular se tratará sobre el desarrollo y la arquitectura dirigida por modelos, el lenguaje de marcado y los lenguajes de transformación.

2.1. EL USO DE MDA. INGENIERÍA DIRIGIDA POR MODELOS

La ingeniería dirigida por modelos (siglas en inglés MDE) es una metodología de desarrollo de software que se basa en la creación de modelos o abstracciones [HONG06] [MONT11]. Tiene como objetivo el aumentar la productividad mediante la compatibilidad entre sistemas, simplificando el proceso de diseño.

Un paradigma de modelado para el MDE se considera eficaz si los modelos son entendidos desde el punto de vista del usuario y pueden servir como base para la implementación de sistemas. Los modelos son desarrollados mediante la comunicación entre los gerentes de producto, diseñadores y miembros del equipo de desarrollo.

Los modelos se construyen para un determinado nivel de detalle o se construyen modelos completos, incluidas las acciones ejecutables. El código puede ser generado a partir de los modelos.

Con la introducción del Lenguaje Unificado de Modelado (UML), MDE se ha vuelto muy popular en la actualidad con un amplio cuerpo de profesionales y herramientas de apoyo. Se han desarrollado distintos tipos de MDE para aportar a la industria distintos estándares para el desarrollo eficiente de aplicaciones. La continua evolución de MDE ha añadido un mayor enfoque en la arquitectura y la automatización.

De acuerdo con Douglas C. Schmidt [SCHM06], las tecnologías MDE ofrecen un enfoque prometedor para hacer frente a la incapacidad de los lenguajes de tercera generación para aliviar la complejidad de las plataformas de desarrollo y expresar conceptos de dominio de una manera eficaz.

A partir de este concepto surge también el concepto de *model-driven development* (MDD), es decir, desarrollo dirigido por modelos [VALL08] [VALL03]. MDD es un paradigma de desarrollo que considera los modelos software como principal elemento del proceso de desarrollo. Normalmente, además, a partir de estos modelos se genera de una forma semi-automática el código.

La iniciativa MDE más conocida es MDA [MILL03a] [MILL03b], ofrecida por el *Object Management Group* (OMG) [CABO09].

2.1.1. Arquitectura dirigida por modelos

2.1.1.1. Definición

MDA responde a las siglas de "*Model Driven Architecture*", que significa "Arquitectura dirigida por modelos". Esta arquitectura fue propuesta por la OMG (*Object Management Group*), consorcio de industria informática sin ánimo de lucro, que tiene como uno de sus principales objetivos el desarrollo de estándares.

El OMG es una organización de compañías de sistemas de información creada en 1990 con el fin de potenciar el desarrollo de aplicaciones orientadas a objetos distribuidas. Esta organización ha definido estándares importantes como UML, CORBA, MOF, entre otros.

En los últimos años, el modelado en el desarrollo de cualquier tipo de software ha tomado mayor interés e importancia, debido a la facilidad que ofrece un buen diseño tanto a la hora de desarrollar como al hacer la integración y mantenimiento de sistemas de software.

Es así que en el año 2001, el OMG definió un marco de trabajo nuevo llamado MDA. La clave del MDA es la importancia de los modelos en el proceso de desarrollo de software. MDA propone la definición y uso de modelos a diferente nivel de abstracción, así como la posibilidad de la generación automática de código a partir de los modelos definidos y de las reglas de transformación entre dichos modelos.

MDE se puede aplicar a la ingeniería del software, sistemas, y datos. La primera herramienta de apoyo a MDE fue CASE (*Computer-Aided Software Engineering*) desarrollada en los ochenta. Con algunas variaciones de las definiciones de modelado se creó lo que se conoce en la actualidad por Lenguaje Unificado de Modelado (UML) [OMG].

MDA es una plataforma para desarrollo de aplicaciones, cuyo objetivo es aumentar la calidad y velocidad de desarrollo de aplicaciones, llevándolo a cabo mediante el aumento de nivel de abstracción, junto al uso de técnicas de modelado, de transformación del modelo y de generación de código.

Los tres principales objetivos de MDA que más tarde se definen son la portabilidad, la interoperabilidad y la reutilización.

2.1.1.2. Fundamentos

MDA defiende la separación de la especificación de la funcionalidad de un sistema y su implementación, independientemente de la plataforma que se utilice.

Por lo tanto, MDA, basado en estándares de la OMG, separa la lógica de negocio y la de la plataforma. A su vez se basa en los principios de abstracción, automatización y estandarización.

La mayoría de las aplicaciones actuales se basan en una arquitectura básica de tres capas (*three-tier*), que son:

- Capa de presentación: la presentación al usuario, con las entradas de datos y las pantallas de consulta.
- Capa de lógica de negocio: donde se procesa la información.
- Capa de datos: el control del almacén de datos.

La arquitectura MDA se centra en el modelo y persigue tres objetivos básicos, todos ellos a través de la separación arquitectónica [GARC11]:

- La portabilidad: la característica que posee un software para ejecutarse en diferentes plataformas. El código fuente del software es capaz de reutilizarse en vez de crearse un nuevo código cuando el software pasa de una plataforma a otra. A mayor portabilidad menor es la dependencia del software con respecto a la plataforma.

- La interoperabilidad: condición mediante la cual sistemas heterogéneos pueden intercambiar procesos o datos. En otras palabras, habilidad que tiene un sistema o producto para trabajar con otros sistemas o productos sin un esfuerzo especial por parte del cliente.

- La reusabilidad: hace referencia a la capacidad del software, en parte o completamente, para ser usado en otro proyecto.

La OMG con esta definición de MDA dispone de herramientas que permiten (véase la Figura 2.1) [OMG] [PELA07] [GRAC10]:

- Especificar sistemas independientes de las plataformas que los soportan, modelando los mismos mediante modelos independientes de plataforma (PIMs, *Platform Independent Models*).
- Escoger la plataforma más adecuada para cada tipo de sistema y especificarla mediante modelos específicos de plataformas (PSMs, *Platform Specific Models*).
- Transformar las especificaciones de los sistemas (los PIMs) a las especificaciones de las plataformas (los PSMs).

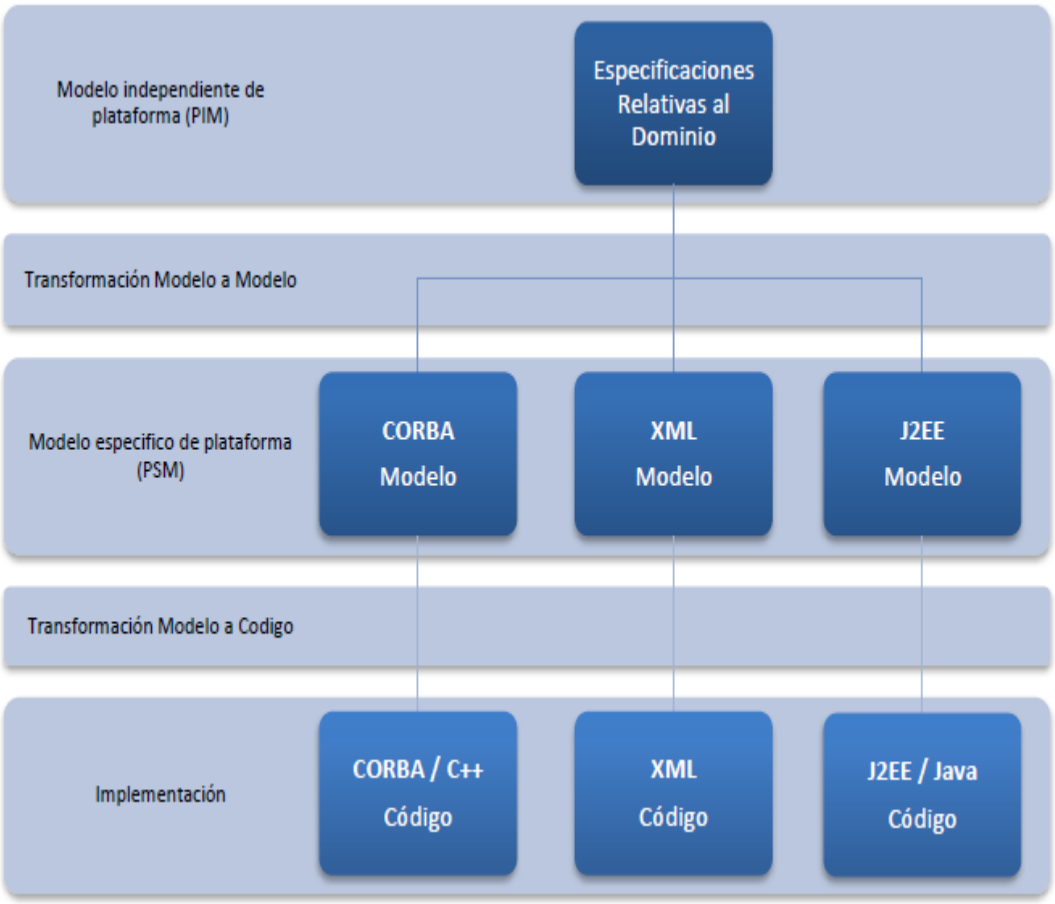


Figura 2.1. Arquitectura dirigida por modelos

La transformación de modelos es fundamental en los diversos enfoques del desarrollo de software dirigido por modelos. Esta tecnología permite caracterizar artefactos capaces de traducir automáticamente modelos que se ajustan a un determinado metamodelo origen, en modelos que se ajustan a un determinado metamodelo objetivo (transformaciones modelo a modelo). Así mismo, estas tecnologías permiten también realizar transformaciones entre modelos y otros formatos de representación (p.ej., entre modelos y formatos textuales, dando lugar a las denominadas transformaciones de modelo a texto y de texto a modelo).

En los últimos años se han definido una gran cantidad de lenguajes de transformación, cada uno con diferentes características. Existen distintas propuestas como lenguajes de transformación entre los que cabe destacar:

- XSLT
- ATL - *a model transformation tecnology* [PELA07] [ECLI]. Lenguaje de transformación de modelos y herramienta para Eclipse desarrollada por el Atlas Group.
- ETL *(Epsilon Transformation Language).* Lenguaje definido sobre la plataforma Epsilon, plataforma desarrollada como un conjunto de plug-ins (editores, asistentes, pantallas de configuración, etc) sobre Eclipse.
- QVT (*Query/View/Transformation*). Especificación estándar de OMG. Está basado en MOF (*Meta Object Facility*) para lenguajes de transformación en MDA [MONT11] [PELA07] [OMG11].

En este proyecto la idea es la de utilizar el lenguaje de transformación XSLT.

2.1.1.3. XSLT

XSLT es un lenguaje estándar de la *World Wide Web Consortium* (W3C) [XSLT] y cuenta con un extenso soporte tecnológico. Las hojas de estilo XSLT - aunque el término de hojas de estilo no se aplica sobre la función directa del XSLT - realizan la transformación del documento utilizando una o varias reglas de plantilla. Estas reglas de plantilla unidas al documento fuente a transformar alimentan un procesador de XSLT, el que realiza las transformaciones deseadas poniendo el resultado en un archivo de salida, o, como en el caso de una página web, las hace directamente en un dispositivo de presentación tal como el monitor del usuario.

XSLT permite transformar un documento XML en otro formato. Este formato puede ser HTML, XML o cualquier otro. En consecuencia, XSLT podría ser utilizado para transformar los modelos, que están representados en el formalismo de XMI.

Como principales características, podemos decir que XSLT tiene una naturaleza híbrida, pues permite tanto construcciones imperativas como declarativas. En su parte declarativa, la aplicación del *pattern-matching* se hace de forma ordenada y recursiva, por lo que XSLT es determinista. Además, desde la perspectiva de metamodelado, efectúa la transformación de grafos, que son árboles, como representación de modelos, y utiliza XPath como lenguaje de consulta. Actualmente, XSLT es muy utilizado ya que permite separar contenido y presentación, aumentando así la productividad.

El lenguaje XSLT tiene sus propias etiquetas, las hay que se utilizan para definir plantillas y hacer llamadas a las mismas, otras para emular estructuras condicionales, de selección, etc. también para funciones como aplicar atributos del fichero XML original. De esta manera, se puede combinar la utilización del archivo origen en UsiXML con un archivo de tipo XSL que haga las veces de plantilla de transformación, para obtener el deseado fichero de la implementación final. De las transformaciones, se pueden obtener dos tipos de ficheros: texto y xml (etiquetado). La forma de realizar transformaciones será totalmente diferente, ya que al estar XSLT basado en XML el texto ha de introducirse mediante etiquetas, es decir, habrá que indicar expresamente que se desea introducir texto. En caso de tener como resultado un archivo etiquetado, se podrán incluir las etiquetas de forma normal [PEÑA07].

A continuación se puede observar la estructura de un documento XSLT, donde se puede observar la plantilla raíz, que es la primera que se ejecutará y que a su vez puede ir aplicando las plantillas que se deseen o aplicando cualquier regla.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0">
  <xsl:template match='/'>
     …
  </xsl:template>
</xsl:stylesheet>
```

Código 2.1. Estructura XSLT

Algunas de las etiquetas más importantes utilizadas en XSLT se encuentran recogidas en la tabla 2.1, junto con una breve descripción de su uso.

Etiqueta	Ejemplo	Descripción
Text	<xsl:text> import </xsl:text>	Introduce texto, necesario cuando el archivo de salida sea en modo texto.
Template	<xsl:template match="/"> …. </xsl:template>	Las plantillas hacen las veces de funciones.
Call-Template	<xsl:call-template name="button"/>	Para hacer llamadas a plantillas.
Value-Of	<xsl:value-of select="@n"/>	Inserta el valor de un atributo del origen.
If	<xsl:if test = "@n =’1’"> <xsl:text>...</xsl:text> </xsl:if>	Sentencia condicional, se realiza lo que contiene si la condición es cierta.
For-each	<xsl:for-each select="box"> … </xsl:for-each>	Hace la acción contenida para cada atributo que cumpla la condición. Especialmente útil en archivos XML ya que podrá existir una o múltiples ocurrencias de una misma etiqueta.
Param	<xsl:param name="n" />	Para recoger parámetros en las plantillas.
With-Param	<xsl:with-param name="n">5</xsl:with-param>	Para pasar parámetros a las plantillas.

Tabla 2.1. Etiquetas XSLT

Relativo a XSLT podemos también resaltar la importancia de XML Schema como lenguaje para definición de *schemas* en el que se expresa el modelo a seguir en las instancias (se describe lo único válido en ellas) utilizando una estructura en árbol con un detallado sistema de tipos y propiedades de orientación a objetos o XSL-FO como lenguaje especializado en la transformación de un documento XML con el formato de presentación final.

XML Schema facilita la descripción de un modelo de dominio gracias a las siguientes características: propiedades de orientación a objetos (herencia, extensibilidad, polimorfismo), capacidad de instanciación, espacios de nombre, riqueza de tipos de datos predefinidos y posibilidades de personalización, expresión formal de relaciones entre elementos, estructura jerárquica en forma de árbol, atributos para recoger las propiedades de los elementos, validación formal de las instancias mediante un *parser* genérico y transformaciones XSLT para exportar o importar información entre modelos.

2.1.1.4. XML

XML, siglas en inglés de *Extensible Markup Language* (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C) [XML]. Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Su uso está en bases de datos, editores de texto, hojas de cálculo, etc. XML es una tecnología sencilla que se integra y complementa con otras. Actualmente tiene un papel importante debido a que permite la compatibilidad entre sistemas para compartir información de una manera segura, fiable y fácil.

Sus principales ventajas son:

- Es extensible: Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan bugs y se acelera el desarrollo de aplicaciones.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.

XML es un meta-lenguaje que permite definir formalmente lenguajes de marcas, de forma que el lenguaje definido sirva para expresar un modelo concreto de información. Se puede sintetizar en 3 niveles la arquitectura de abstracción XML (ver figura 2.2):

• Lenguaje genérico de XML Schema. Constituye un meta-modelo de datos. Permite describir descripciones de datos. Define el meta-lenguaje que puede emplearse para construir lenguajes específicos.

• Lenguaje particular definido en un *schema* XML. Constituye un modelo de datos. Permite describir datos. Define el lenguaje que se empleará para describir ciertos datos particulares.

• Documento XML. Descripción de datos en base al lenguaje definido por el *schema*.

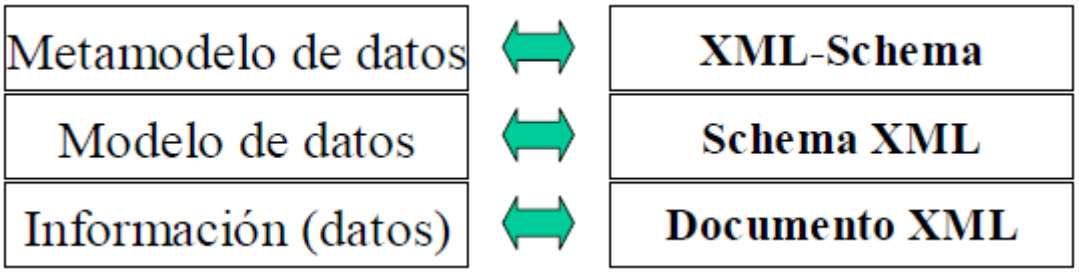


Figura 2.2. Relación modelo - XML

2.1.1.5. HTML5

HTML5 (*HyperText Markup Language*, versión 5) es la quinta revisión importante del lenguaje básico de *la World Wide Web* [HTML], HTML. HTML5 especifica dos variantes de sintaxis para HTML: HTML (text/html), la variante conocida como HTML5 y una variante XHTML conocida como sintaxis XHTML5 que deberá ser servida como XML (XHTML) (application/xhtml+xml). HTML y XHTML por primera vez se han desarrollado en paralelo.

Todavía se encuentra en modo experimental, lo cual indica la misma W3C; aunque ya es usado por múltiples desarrolladores web por sus avances, mejoras y ventajas.

Al no ser reconocido en versiones más antiguas de navegadores por sus nuevas etiquetas, se le recomienda al usuario actualizar a la versión más actual. El desarrollo de este código es regulado por el Consorcio W3C. HTML5 añade etiquetas para manejar la Web Semántica (Web 3.0): header,footer, article, nav, time(fecha del contenido), link rel=‘’ (tipo de contenido que se enlaza) que permiten describir cual es el significado del contenido. Por ejemplo su importancia, su finalidad y las relaciones que existen. No tienen especial impacto en la visualización, se orientan a buscadores. Los buscadores podrán indexar e interpretar esta meta información para no buscar simplemente apariciones de palabras en el texto de la página. Permite incorporar a las páginas ficheros RDF / OWL (con meta información) para describir relaciones entre los términos utilizados. Empresas como Google, Microsoft o Apple, entre, han manifestado ya su apoyo a HTML5.

HTML5 establece una serie de nuevos elementos y atributos [KJAE09] que reflejan el uso típico de los sitios web actuales. Algunos de estos cambios son técnicamente similares a las etiquetas y , pero tienen un significado

semántico, como por ejemplo <nav> (bloque de navegación del sitio web) y <footer>. Otros elementos proporcionan nuevas funcionalidades a través de una interfaz estandarizada, como los elementos <audio> y <video>.

Existen mejoras en el elemento <canvas>, capaz de renderizar en algunos navegadores elementos 3D.

Algunos elementos de HTML 4.01 han quedado obsoletos, incluyendo algunos puramente de presentación, como <font> y , cuyos efectos son manejados por las hojas de estilo (CSS). También se resalta la importancia del scripting DOM para el comportamiento de la web.

Cuando los navegadores renderizan una página, construyen un "Modelo de Objeto de Documento" (*Document Object Model - DOM*), una colección de objetos que representan los elementos del HTML en la página. Cada elemento (<p>, , , etc.) es representado en el DOM por un objeto diferente.

Todos los objetos DOM comparten unas características comunes, aunque algunos poseen más que otros. En los navegadores que soportan rasgos del HTML5, algunos objetos tienen una única propiedad y observando al DOM se puede saber las características que soporta el navegador.

Con el tránsito definitivo a HTML5, algunos elementos dejarán de existir, entre los que cabe destacar:

- <acronym> Se usaba para describir acrónimos (por ejemplo AVE) en HTML4, sin embargo ahora deberemos usar la etiqueta <abbr>.
- <applet> Esta etiqueta se usaba para introducir applets dentro de la web, sin embargo, ahora debemos usar <object>.
- Centraba horizontalmente el contenido. La solución correcta es usar CSS.
- <big> Servía para hacer el texto más grande, mediante CSS podemos hacer esto mismo de varias maneras.
- <basefont> Definía características de la fuente por defecto: color, tamaño y familia; para todo el documento. Esto mismo podemos hacerlo mediante CSS.
- <font> Especificaba la fuente de un texto, debe evitarse su uso. Su labor debe hacerse usando CSS.
- <frame> Cuando no existía AJAX y no se podían recargar los de manera asíncrona, los *frames* eran una de las mejores formas de hacer recargas parciales de las páginas. Esta opción desaparece en HTML5.
- <frameset> Relacionada con la etiqueta <frame>, era parte del funcionamiento de los frames.
- <noframes> También relacionada con las dos anteriores.
- <u> Se utilizaba para indicar texto subrayado, ahora se realiza mediante CSS, por ejemplo: text-decoration:underline.

Además todos los atributos referentes a la presentación han sido eliminados, debido a que las hojas de estilo (CSS) son las que deben de cubrir la parte de diseño.

Como se ha visto anteriormente HTML5 también añade nuevas etiquetas.

En HTML5 existen varios elementos que sirven para estructurar de una manera más razonable una página web, estableciendo qué es cada sección, y reemplazando en muchas ocasiones las etiquetas . Estos son los elementos:

- <section> representa una sección "general" dentro de un documento o aplicación, como un capítulo de un libro. Puede contener subsecciones.
- <article> representa un contenido independiente en un documento, el caso más claro son las entradas de un blog o las noticias de un periódico online.
- <aside> representa un contenido que está muy poco relacionado con el resto de la página, como una barra lateral. Permite, por ejemplo, delimitar el contenido más importante de un contenido de apoyo o de segundo nivel.
- <header> representa la cabecera de una sección.
- <footer> representa el pie de una sección, con información acerca de la página/sección, como el autor, el copyright o el año.
- <nav> representa una sección dedicada a la navegación entre el sitio.

Figura 2.3. Estructura HTML5 versus Estructura actual

En la anterior imagen vemos un ejemplo de cómo cambiaría un documento escrito en HTML normal a HTML5 con estos elementos.

El elemento <input> ha sido ampliado y ahora permite estos tipos de datos:

- datetime, datetime-local, date, month, week, time, para indicar una fecha/hora.
- number para que el usuario indique un número.
- range para indicar un rango entre dos números.
- email para indicar un correo electrónico.
- url para indicar una dirección web.
- search para indicar una búsqueda.
- color para indicar un color.

Otros elementos importantes:

- <audio> y <video> sirven para incrustar un contenido multimedia de sonido o de vídeo, respectivamente. Esto permite reproducir/controlar vídeos y audios sin necesidad de plugins como el de Flash. Por ejemplo, el portal de vídeo Youtube actualmente permite el uso en versión de pruebas de un reproductor de video en HTML5 con el códec de vídeo h.264 o con el formato WebM.
- <embed> sirve para contenido incrustado pero no nativo, sino ejecutado por plugins como el de Flash.
- <canvas> es un elemento complejo que permite generar gráficos, dibujando elementos dentro de él.

Los nuevos estándares abiertos de la plataforma web como CSS3, HTML5, JavaScript proporcionan herramientas y características para construir aplicaciones web más elaboradas e interactivas.

PARTE III

Desarrollo de la Investigación

Capítulo 3
Desarrollo de la investigación

En este capítulo se expone el desarrollo de la aplicación, diseño, análisis de requisitos e implementación.

3.1. ANÁLISIS DE REQUISITOS

El presente documento cubre la fase de requisitos del sistema que se va a desarrollar. Para ello se detallarán los objetivos y el alcance del sistema así como la especificación de requisitos.

3.1.1. Definición del sistema de información

A continuación, se definen las características de la aplicación sistema y los distintos objetivos que persigue.

3.1.1.1. Alcance del sistema

Este proyecto pretende la creación de una aplicación para la generación de webs de guías de turismo aplicando una arquitectura dirigida por modelos. Una aplicación que permita realizar de forma fácil y cómoda la generación de una guía turística desarrollada en HTML5.

3.1.1.2. Objetivos

El proyecto pretende permitir a los usuarios poder generar guías turísticas en formato Web (HTML5) de forma sencilla y rápida.

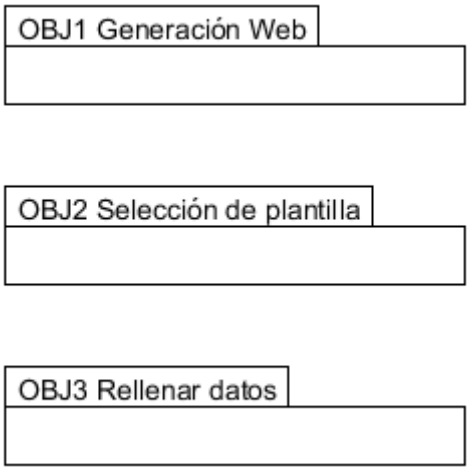


Figura 3.1. Objetivos aplicación

OBJ1 Generación Web
Generación de código HTML

Tabla 3.1. OBJ1 Generación Web

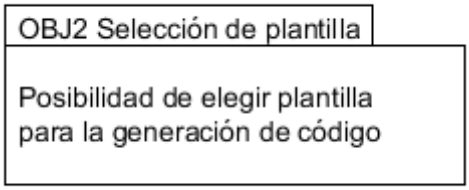


Tabla 3.2. OBJ2 Selección de plantilla

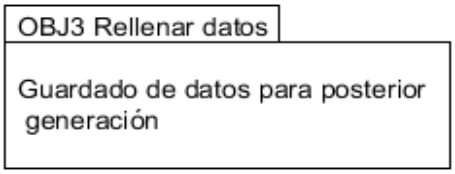


Tabla 3.3. OBJ3 Rellenar datos

3.1.1.3. Glosario de términos

- MDA - (*Model Driven Architecture*) Arquitectura dirigida por modelos.
- UWE - UML- *based Web Engineering*.
- XML - (*Extensible Markup Language*) Lenguaje de marcas extensible.
- XSL - (*Extensible Stylesheet Language*) Lenguaje extensible de hojas de estilo.
- XSLT - (*Extensible Stylesheet Language Transformatios*) Transformaciones XSL

3.1.2. Catálogo de requisitos

El catálogo de requisitos identifica con un alto nivel de abstracción los diferentes aspectos que deberá cubrir la aplicación.

3.1.2.1. Requisitos de almacenamiento de información

Los requisitos de almacenamiento de información son aquellos que definen a grandes rasgos los datos que deberá almacenar internamente el sistema.

El sistema almacenará los criterios necesarios para la generación de páginas Web. Son los siguientes:

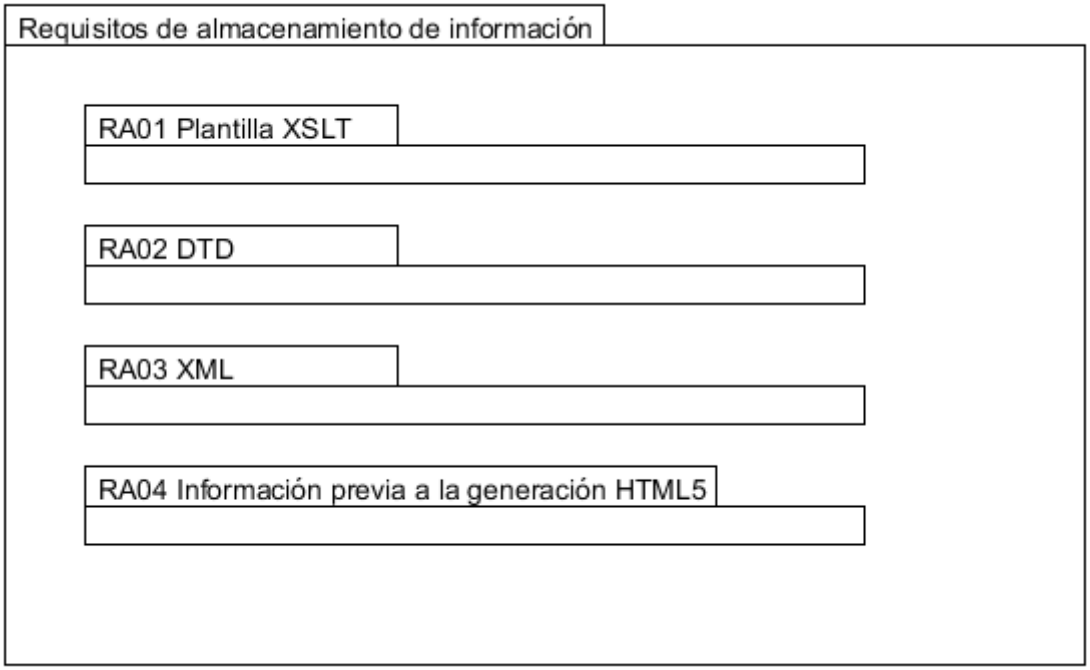


Figura 3.2. Requisitos de almacenamiento de información

3.1.2.2. Requisitos funcionales: funcionalidad del sistema

Los requisitos funcionales del presente sistema son los siguientes:

- Creación de guías.
- Selección de plantillas.
- Validación de datos.
- Exportación de datos.

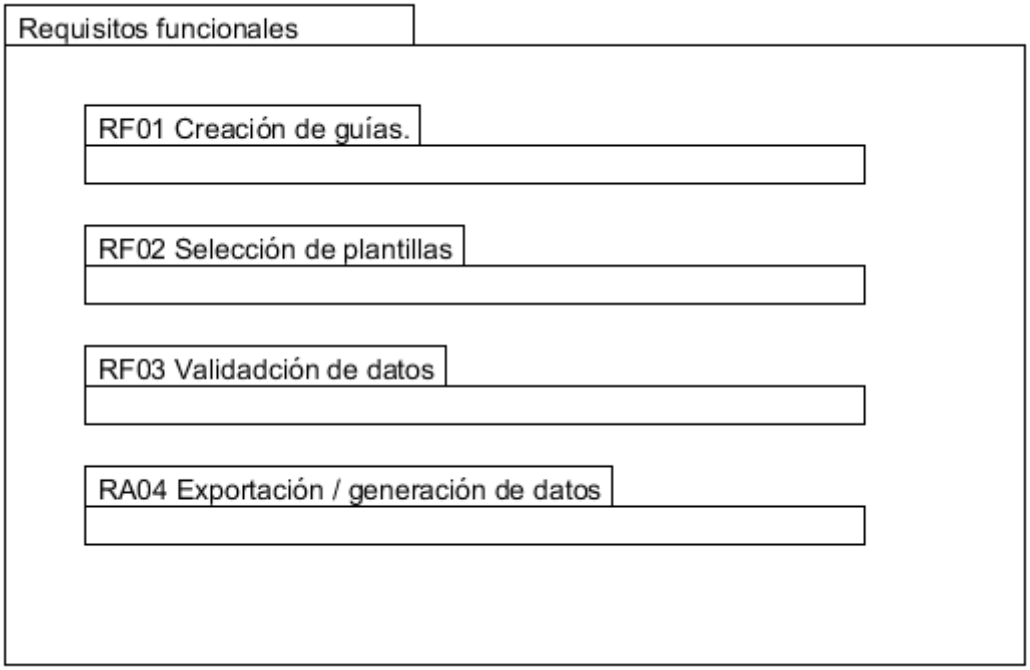


Figura 3.3. Requisitos funcionales

Diagramas de caso de uso:

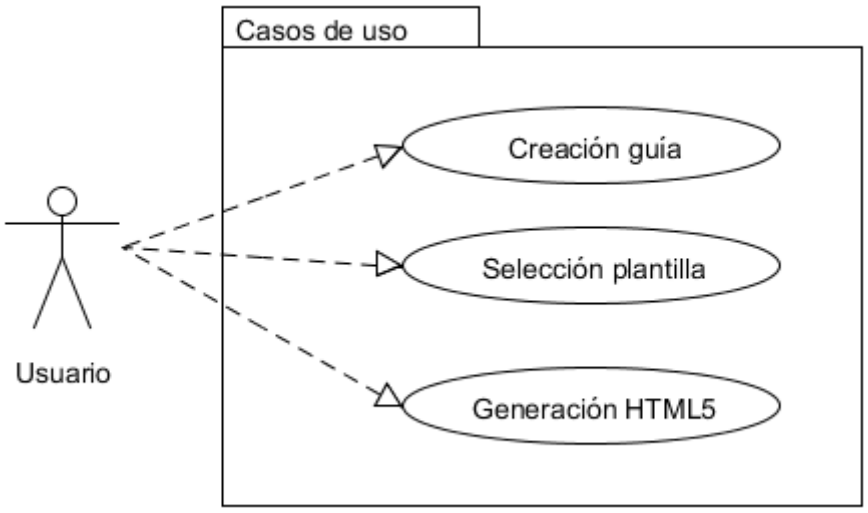


Figura 3.4. Casos de uso

3.1.2.3. Requisitos no funcionales

Los requisitos no funcionales son aquellos que definen propiedades globales a toda la aplicación, orientados fundamentalmente a la calidad del sistema.

- Transacciones: el sistema deberá manejar transacciones en las operaciones de alta.

- Entorno tecnológico: el sistema deberá ejecutarse en un entorno Windows independientemente del modo en que el procesador administre la información. Por lo tanto, la aplicación deberá poder ejecutarse tanto en entornos de 32 bits como de 64 bits.
- Fiabilidad: el sistema deberá escalar de forma sencilla y se diseñará para soportar el volumen de transacciones requerido.
- Extensibilidad: el sistema debe ser diseñado de modo que admita nuevas extensiones de funcionalidad para cubrir las necesidades específicas futuras.

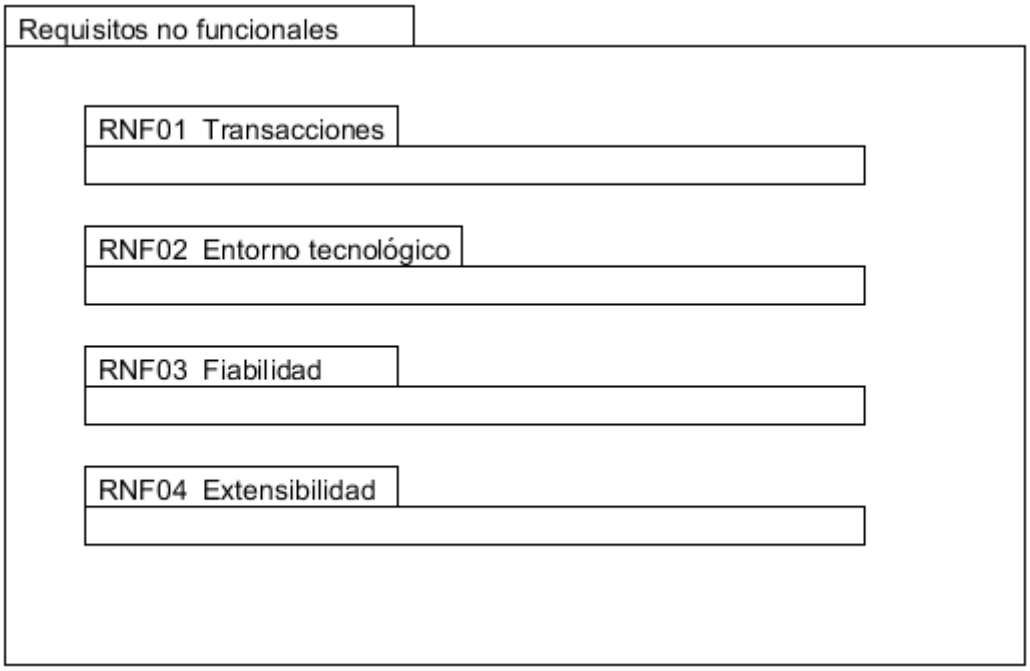


Figura 3.5. Requisitos no funcionales

3.1.3. Definición de actores

ACT01	Usuario
Descripción	Este actor representa al usuario de la aplicación

Tabla 3.4. Definición de actores

3.1.4. Casos de uso UWE

3.1.4.1. Navegación

El objetivo del modelo de navegación es representar los nodos y enlaces de la aplicación y el diseño de rutas de navegación.

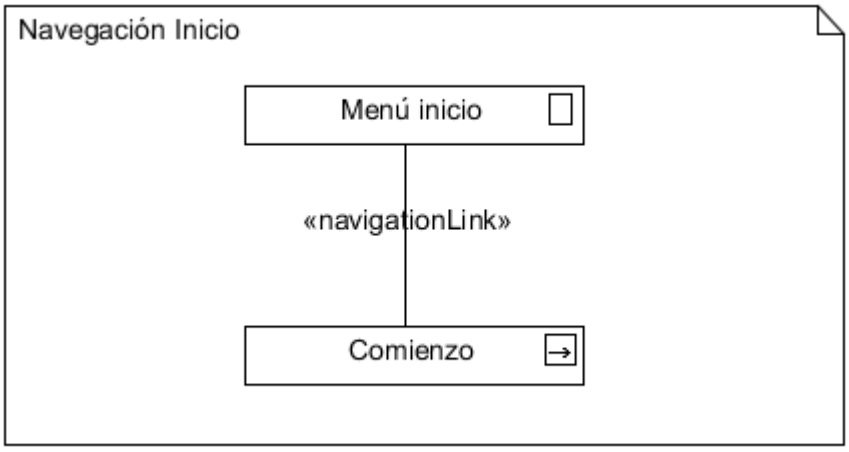


Figura 3.6. Navegación Inicio

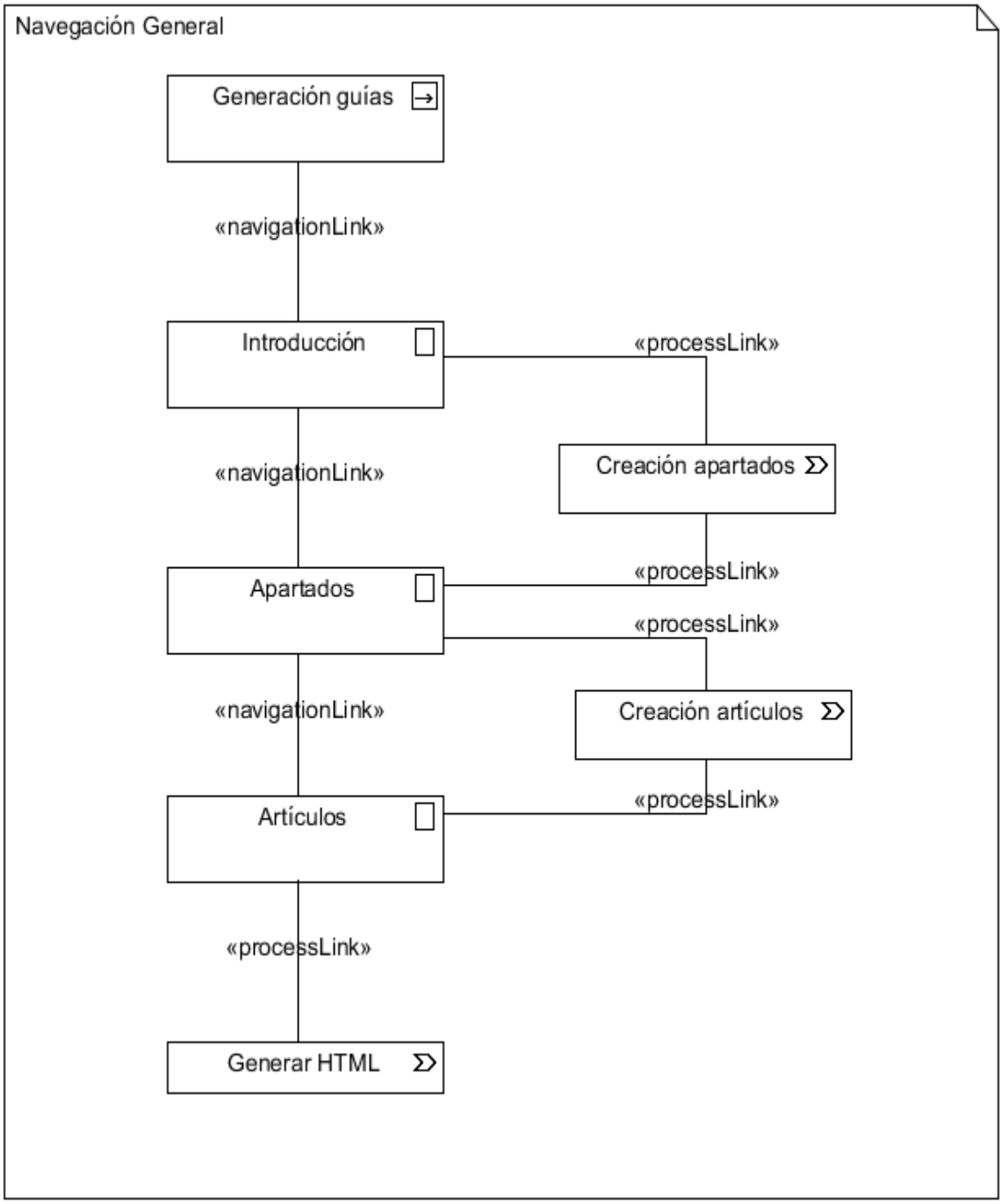


Figura 3.7. Navegación General

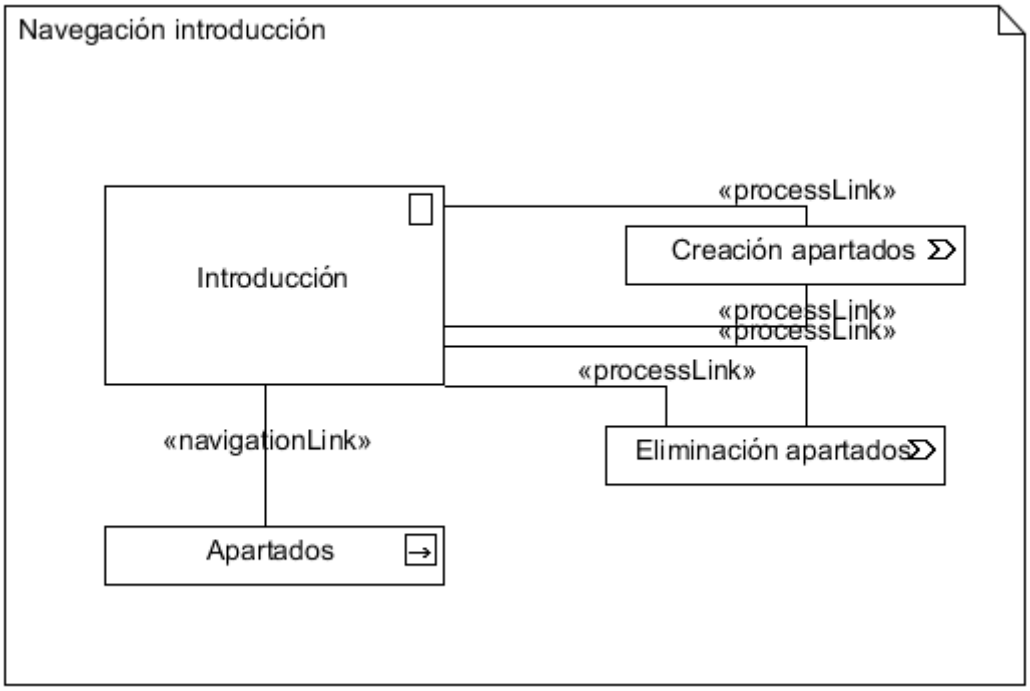


Figura 3.8. Navegación Introducción

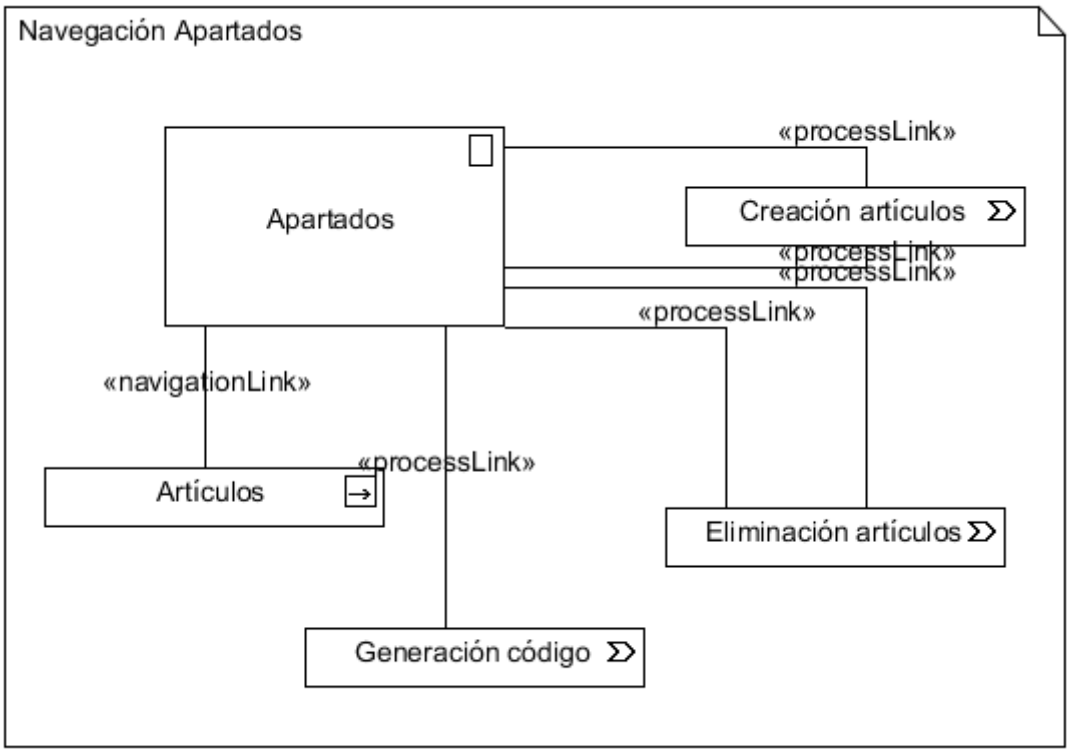


Figura 3.9. Navegación Apartados

3.1.4.2. Presentación

El objetivo del modelado de presentación es representar las páginas web o una parte de ellas, es una alternativa a realizar bocetos o la implementación de un prototipo.

Figura 3.10. Presentación Inicio

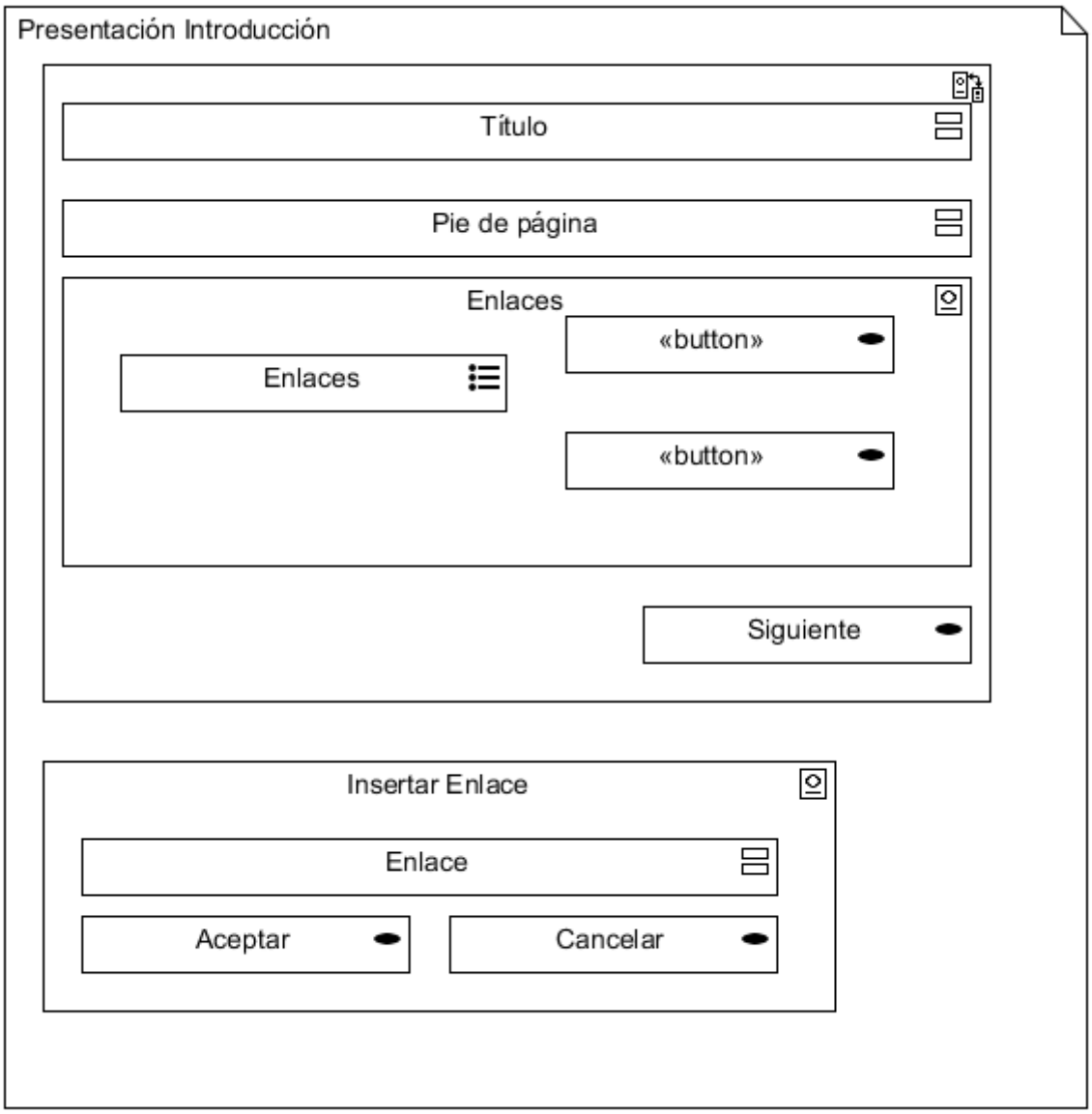


Figura 3.11. Presentación Introducción

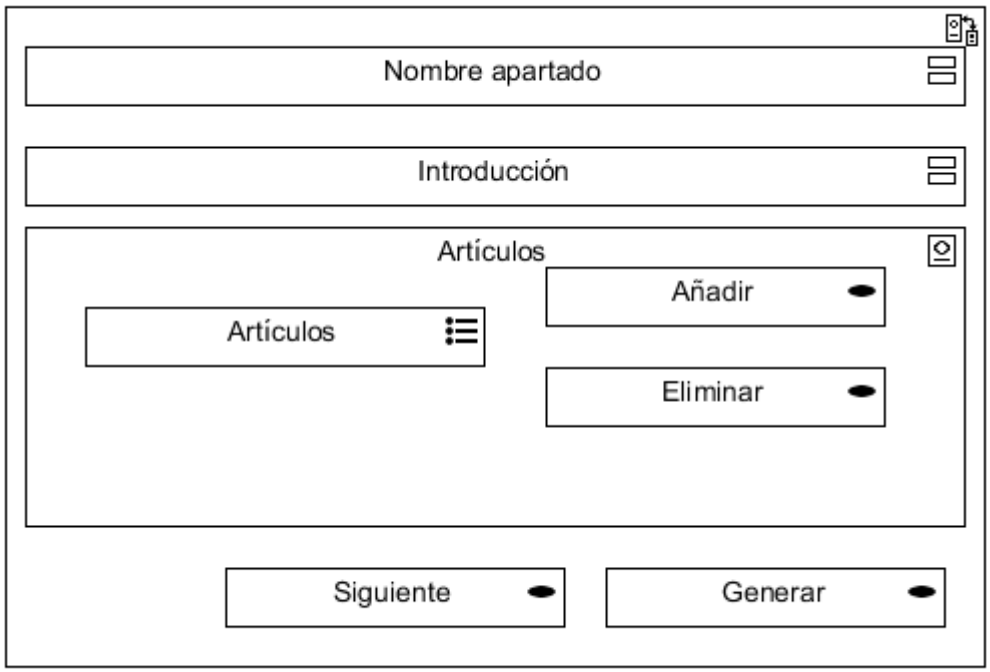


Figura 3.12. Presentación Apartado

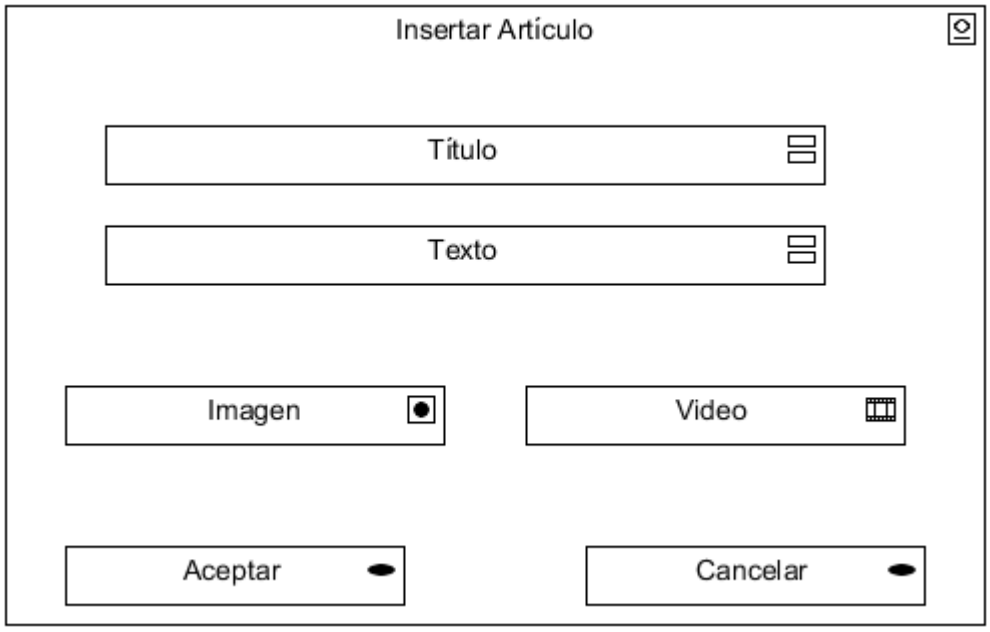


Figura 3.13. Presentación Artículos

Figura 3.14. Presentación Validaciones

3.2. DISEÑO

El presente documento define la arquitectura general del sistema de información, especificando las distintas particiones físicas del mismo, la descomposición lógica en subsistemas de diseño y la ubicación de cada subsistema en cada partición, así como la especificación detallada de la infraestructura tecnológica necesaria para dar soporte al sistema de información.

3.2.1. Definición de la arquitectura del sistema

En esta sección se realiza una descripción formal de la arquitectura del sistema, en la que se incluyen sus componentes, las relaciones entre sí y el medio ambiente, tanto física como lógica.

3.2.1.1. Arquitectura física

La arquitectura física del sistema que se está tratando se representa mediante el siguiente diagrama de componentes hardware.

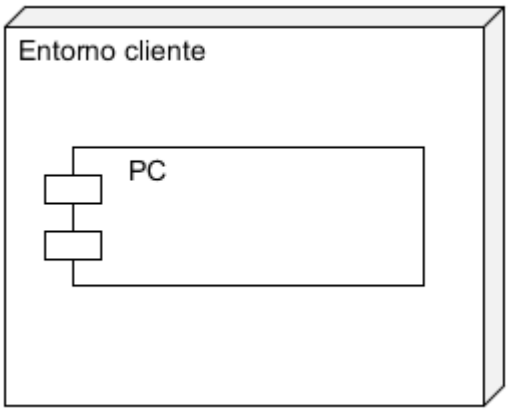


Figura 3.15. Arquitectura física

Entorno cliente

Aquí se detallan los componentes de cliente que forman parte de la arquitectura física del sistema.

- PC: Esta máquina representa a los ordenadores personales desde la cual acceder al sistema desarrollado.

3.2.1.2. Arquitectura lógica

La arquitectura lógica del sistema que se está tratando se representa mediante el siguiente diagrama de componentes software.

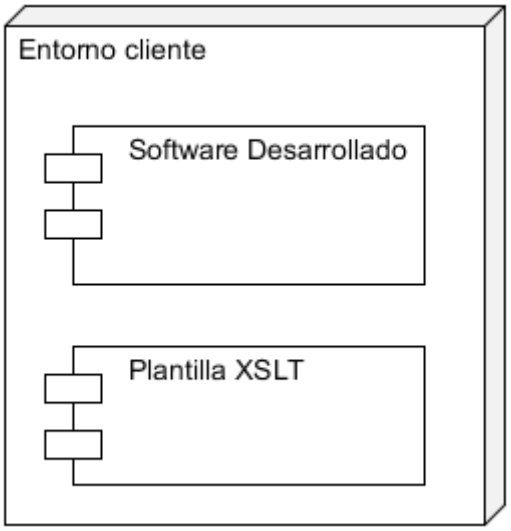


Figura 3.16. Arquitectura lógica

Entorno cliente

A continuación, se detallan los componentes de cliente que forman parte de la arquitectura lógica del sistema.

- Software Desarrollado
  Hace referencia al software desarrollado para cumplir los requisitos de la aplicación.
- Plantilla XSLT
  Este componente hace referencia al fichero XSLT necesario para que la aplicación desarrollada genere el correspondiente documento HTML.

3.2.2. Diseño arquitectónico

La arquitectura de diseño especifica la forma en que los artefactos software de más bajo nivel interactúan entre sí para realizar una determinada funcionalidad de usuario y de sistema.
Se puede ver en el siguiente diagrama (sigue el patrón Modelo Vista Controlador (MVC)):

3.2.2.1. Capa de presentación

- Clases C#
  Representa a las clases Java responsables del control de la navegación a través de los formularios y de las operaciones de lógica de negocio a efectuar.
  Este grupo de artefactos software forman la capa de presentación del sistema (se compone de los elementos de la vista y los elementos de control de la misma).

- Formularios
  Representa a los formularios Windows Forms, que muestran la información al usuario.

3.2.2.2. Capa de negocio

Este grupo de artefactos software forman la capa de negocio del sistema (se compone de los elementos del modelo de dominio y los servicios).

- Clases de servicios
  Representa las clases que implementan la lógica de negocio del sistema.
- Clases del dominio
  Representa las clases de análisis implementadas en Visual C#.

3.2.2.3. Capa de persistencia

Este grupo de artefactos software forman la capa de integración del sistema (clases de abstracción para el acceso a datos y lógica incrustada).

- Clases de acceso a datos
  Representa las clases que proporcionan una capa de abstracción entre la aplicación y las plantillas XSLT.

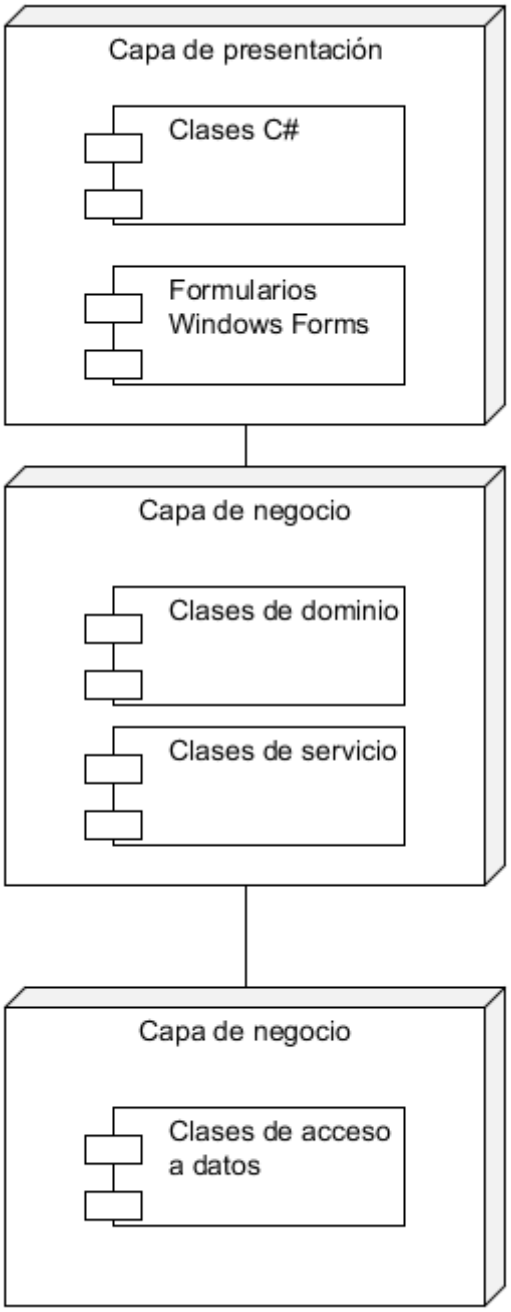


Figura 3.17. Diseño arquitectónico

3.3 HERRAMIENTAS UTILIZADAS

En este apartado se detallan qué herramientas han sido utilizadas para el buen desarrollo del proyecto. Las herramientas se han dividido en tres secciones diferenciadas, herramientas usadas en el diseño y análisis, herramientas usadas en la implementación y por último las herramientas usadas para medir y controlar la calidad del software desarrollado.

3.3.1. Herramientas para el diseño y análisis

Para este apartado se ha usado UMLet [UMLE] que es una herramienta *open-source* para creación de diagramas UML. Esta aplicación permite realizar diagramas UWE. También da la posibilidad de crear los principales tipos de diagramas (casos de uso, entidad - relación, etc.). El formato nativo de la herramienta es UXF (UML eXchange Format) pero también permite la exportación de los diagramas a imágenes (.eps, .jpj, .png), vectores (.svg) o documento (.pdf). Puede ser ejecutado en distintos sistemas operativos ya que es posible su ejecución mediante la máquina virtual de Java.

3.3.2. Herramientas para la implementación

Las herramientas para el desarrollo del software han sido las siguientes, por orden de uso:

3.3.2.1. XmlSpy

Altova XMLSpy [XMLSa] es un editor de XML y entorno de desarrollo para el modelado, edición, transformación y depuración de tecnologías XML-relacionadas. Ofrece, entre otras características, un generador de código, convertidores de archivos, depuradores, perfiladores, la integración de base de datos completa, soporte para XSLT, XPath, XQuery, WSDL, SOAP, XBRL u Office Open XML (OOXML) de los documentos.

XMLSpy abstrae la complejidad de trabajar con tecnologías basadas en XML a través de su interfaz de usuario.

Esta aplicación proporciona el cumplimiento de los últimos estándares, como XML, DTD, Schema XML, XSLT 1.0 y 2.0, 1.0 y 2.0 XPath, XQuery y XML Signature, así como SOAP y WSDL 1.1 / 2.0 para desarrollo de servicios Web. El menú XSL/Xquery nos permite realizar transformaciones XSL con un XML asociado.

Las funciones de edición inteligente de XMLSpy como la finalización automática, los ayudantes de entrada, coloreado de sintaxis, asistentes, depuradores y perfiladores hacen más simple el desarrollo de XML, XSLT y Schema XML [XMLSb].

XLM Spy es una herramienta de pago, pero puede descargarse una versión de prueba de 30 días de duración.

3.3.2.2. Microsoft Visual Studio Express C#

Para desarrollar una aplicación .NET sólo es necesario instalar el .NET Framework 4.0, el bloc de notas y una ventana de línea de comandos. Sin embargo, lo recomendable es utilizar un ambiente integrado de desarrollo (IDE).

Microsoft Visual C# 2010 Express es un IDE totalmente gratuito que podemos utilizar para empezar a desarrollar aplicaciones .NET utilizando el lenguaje de programación C#.

Es un aplicación para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Visual C# 2010 Express es parte de la familia Visual Studio 2010 Express, un conjunto de herramientas que los desarrolladores de Windows de cualquier nivel pueden utilizar para crear aplicaciones personalizadas mediante configuración básica o experta. Visual C# está diseñado para la construcción de una gran variedad de aplicaciones que se ejecutan en el .NET Framework.

Microsoft ofrece gratuitamente las Express Editions, que son iguales al entorno de desarrollo comercial pero sin características avanzadas.

3.3.2.3. C#

C# es un lenguaje de programación simple pero eficaz, diseñado para escribir aplicaciones empresariales.

El lenguaje C# es una evolución de los lenguajes C y C++. Utiliza muchas de las características de C++ en las áreas de instrucciones, expresiones y operadores.

C# presenta considerables mejoras e innovaciones en áreas como seguridad de tipos, control de versiones, eventos y recolección de elementos no utilizados (liberación de memoria).

C# proporciona acceso a los tipos de API más comunes: .NET Framework, COM, Automatización y estilo C [ALBA10] [BELL02]. Asimismo, admite el modo unsafe, en el que se pueden utilizar punteros para manipular memoria que no se encuentra bajo el control del recolector de elementos no utilizados.

Entre las características que podemos encontrar en este lenguaje (algunas propias de la plataforma .NET) destacan las siguientes:

- Orientación a objetos: C# es un lenguaje orientado a objetos. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código y

datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

- Orientado a componentes: la propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular. La sintaxis de C# incluye por ejemplo formas de definir propiedades, eventos o atributos.

- Gestión automática de memoria: esto tiene el efecto en el lenguaje de que no es necesario incluir instrucciones de destrucción de objetos. Sin embargo, dado que la destrucción de los objetos a través del recolector de basura es indeterminista y sólo se realiza cuando éste se active, C# también proporciona un mecanismo de liberación de recursos determinista a través de la instrucción using.

- Seguridad de tipos: C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que permite evitar que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto. Para ello se toman medidas del tipo: no se pueden usar variables no inicializadas; sólo se admiten conversiones entre tipos compatibles; se puede controlar la producción de desbordamientos en operaciones aritméticas, etc.

- Sistema de tipos unificado: todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada System.Object

3.3.3. Herramientas para medir y controlar la calidad

3.3.3.1. FxCop

FxCop es una herramienta que ayuda a mejorar la calidad de las aplicaciones y librerías desarrolladas en cualquier versión de .Net, analizando de forma automática los ensamblados desde distintas perspectivas y sugiriendo mejoras cuando detecta algún problema o incumplimiento de las pautas de diseño para desarrolladores de librerías para .Net Framework [FXCOa].

En cualquiera de los dos casos, el análisis de los ensamblados se realiza sometiéndolos a la comprobación de una serie de reglas basadas en buenas prácticas y consejos para asegurar la robustez y mantenibilidad de código. Del resultado del mismo obtendremos un informe con advertencias agrupadas en las siguientes categorías:

- Advertencias de diseño: recoge avisos de incumplimientos de buenas prácticas de diseño para .Net Framework, como pueden ser uso de constructores públicos en tipos abstractos, interfaces o namespaces vacíos, uso de parámetros out, capturas de excepciones genéricas, etc.

- Advertencias de globalización: avisan de problemas relacionados con la globalización de aplicaciones y librerías, como puede ser el uso de aceleradores de teclado duplicados, inclusión de rutas a carpetas de sistema dependientes del idioma ("archivos de programa"), etc.

- Advertencias de interoperabilidad: analiza problemas relativos al soporte de interacción con clientes COM, como el uso de tipos auto layout visibles a COM, utilización de System.Int64 en argumentos (que no pueden ser usados por clientes VB6), o la sobrecarga de métodos.

- Advertencias de movilidad: cuestionando el soporte eficiente de características de ahorro de energía, como uso de procesos con prioridad ProcessPriorityClass.Idle. o inclusión de Timers que se repitan más de una vez por segundo.

- Advertencias de nombrado: detectan las faltas de cumplimiento de las guías y prácticas recomendadas en cuanto al nombrado de elementos (clases, métodos, variables, etc.), como uso de nombres de parámetros que coinciden con nombres de tipo, y más con los propios de un lenguaje concreto, mayúsculas y minúsculas no utilizadas correctamente, eventos que comiencen por "Before" o "After", puesto que deben nombrarse conjugando verbos en función del momento que se producen (p.e., Closing y Closed en lugar de BeforeClose y AfterClose), y un largo conjunto de comprobaciones.

- Advertencias de rendimiento: ayudan a detectar problemas en el rendimiento de la aplicación o librería, comprobando puntos como el número de variables locales usadas, la existencia de miembros privados o internos (a nivel de ensamblado) no usados, creación de cadenas (strings) innecesarias, por llamadas múltiples a ToLower() o ToUpper() sobre la misma instancia, realización de conversiones (castings) innecesarios, concatenaciones de cadenas en bucles, etc.

- Advertencias de portabilidad: recoge observaciones interesantes para la portabilidad a distintas plataformas, como el uso de declaraciones PInvoke.

- Advertencias de seguridad: se centran en analizar aspectos que podrían dar lugar a aplicaciones o librerías inseguras, avisando de problemas potenciales como la ausencia de directivas de seguridad, punteros visibles o uso de arrays de sólo lectura, entre otros.

- Advertencias de uso: analizan el uso apropiado del framework .Net realizando multitud de chequeos sobre el código, detectando aspectos como ausencia la liberación (dispose) explícita de tipos IDisposable, resultados de métodos no usados, uso incorrecto de NaN, etc.

3.4 IMPLEMENTACIÓN

Las herramientas que se necesitan para la construcción de nuestro sistema se han nombrado anteriormente en el apartado 3.3 y en el presente se describe brevemente cómo ponerlas en funcionamiento. Además se realiza una explicación básica de las partes del código que conforman la aplicación. Por supuesto, si se quiere entrar en mayor detalle sobre la implementación de la aplicación bastará con ver su código incluido en el CD del proyecto, sin embargo, se han introducido algunos fragmentos de código para intentar dar mayor claridad a las explicaciones.

Para poder empezar a generar el código de nuestro sistema se debe preparar todo el entorno. Cabe destacar que en el caso de software que sea de uso común y existan una gran cantidad de recursos en la web, nos remitiremos a dar enlaces en los que consultar la instalación por si se necesitara, pero no se detalla en esta memoria todo el proceso necesario para llevarla a cabo.

Los pasos de preparación del equipo son los siguientes:

- Instalación de Microsoft Visual C# 2010 Express para el desarrollo de la aplicación de escritorio.
- Instalación de Altova XMLSpy Enterprise 2012. En este caso se ha instalado la release 2.0 en su versión de prueba. Su instalación no es del todo necesaria pero si de gran ayuda como se ha visto en apartados anteriores.

La estructura de código y carpetas se pueden ver en el siguiente esquema:

- [raiz]: En la raíz del proyecto se encuentran los formularios de Windows Form.
- [librerías]: Esta carpeta contiene el código de negocio necesario para el correcto funcionamiento de la aplicación.
- [Resources]: En esta carpeta se incluye el XML Schema y el XSLT por defecto de la aplicación.

Las piezas fundamentales para el desarrollo de aplicaciones de escritorio haciendo uso de Visual C# son los formularios. Un formulario es, en última instancia, una hoja en blanco que el desarrollador rellena con controles, para crear una interfaz de usuario, y con código, para procesar los datos. Los controles son objetos contenidos en objetos de formulario. Cada tipo de control tiene su propio conjunto de propiedades, métodos y eventos que lo hacen adecuado para un propósito en particular. Es posible manipular los controles y escribir código para agregar controles dinámicamente, en tiempo de ejecución.

A continuación se muestra un ejemplo de creación de formularios mediante el archivo Principal.cs así como la propia pantalla:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Schema;
using System.IO;
using System.Xml.Xsl;
using System.Xml.Linq;
using System.Collections.ObjectModel;
using WindowsFormsApplication1.Librerias;

namespace WindowsFormsApplication1
{
     public partial class FrmPrincipal : Form
     {
     private static string cadenaXml = string.Empty;
     private static XElement montar;

     public static string CadenaXml
     {
          get { return cadenaXml; }
          set { cadenaXml = value; }
     }

     public static XElement Montar
     {
          get { return montar; }
          set { montar = value; }
     }

     public FrmPrincipal()
     {
          InitializeComponent();
     }

     private void frmPrincipal_FormClosing(object sender,
FormClosingEventArgs e)
     {
          Application.Exit();
     }

     private void lstCap_SelectedIndexChanged(object
sender, EventArgs e)
     {
```

```
     }

     /// <summary>
     /// Muestra dialogo para añadir un apartado
     /// </summary>
     /// <param name="sender"></param>
     /// <param name="e"></param>
     private void btnMas_Click(object sender, EventArgs e)
     {
          Dialogo dialogo = new Dialogo();
          //Muestra popup para añadir apartado
          if (dialogo.ShowDialog() == DialogResult.OK)
          {
               if (!lstCap.Items.Contains(dialogo.Apartado)
&& !string.IsNullOrEmpty(dialogo.Apartado))
               {
                    lstCap.Items.Add(dialogo.Apartado);

               }
          }
          Collection<string> apa = Apartados;
     }

     public Collection<string> Apartados
     {
          get
          {
               Collection<string> lista = new
Collection<string>();
               foreach (Object o in lstCap.Items)
               {
                    lista.Add(o.ToString());
               }
               return lista;
          }
          set
          {
               Apartados = value;
          }
     }

     private void btnSiguiente_Click(object sender,
EventArgs e)
     {
          if (validar())
          {
               MontarXml();
               Apartado apartado = new Apartado();
               apartado.ApartadosArticulos = Apartados;
               apartado.Montar = montar;
```

```
                apartado.FormPrincipal = this;
                apartado.Show();
                this.Hide();
           }
      }

      /// <summary>
      /// Montaje de enlaces, titulo y pie en xmml
      /// </summary>
      public void MontarXml()
      {
           montar = new XElement("Web", new
XAttribute("Titulo", txtTitulo.Text));
           XElement aptXml = new XElement("Enlaces");
           foreach (String apt in Apartados)
           {
                aptXml.Add(new XElement("Enlace", apt));
           }
           montar.Add(aptXml);
           montar.Add(new XElement("Pie", txtPie.Text));
      }

      /// <summary>
      /// Validaciones antes de continuar
      /// </summary>
      /// <returns></returns>
      bool validar()
      {
           if (string.IsNullOrEmpty(txtTitulo.Text))
           {
                MessageBox.Show("Debe introducir un título",
"Mensaje", MessageBoxButtons.OK,
MessageBoxIcon.Information,
MessageBoxDefaultButton.Button1,
MessageBoxOptions.RightAlign);
                return false;
           }
           if (lstCap.Items.Count == 0)
           {
                MessageBox.Show("Debe de introducir al menos
un apartado", "Mensaje", MessageBoxButtons.OK,
MessageBoxIcon.Information,
MessageBoxDefaultButton.Button1,
MessageBoxOptions.RightAlign);
                return false;
           }
           return true;
      }
```

```
      /// <summary>
      /// Elimina un apartado de la lista
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void btnMenos_Click(object sender, EventArgs
e)
      {
            if (lstCap.SelectedIndex >= 0)
            {
                  Apartados.RemoveAt(lstCap.SelectedIndex);
                  lstCap.Items.RemoveAt(lstCap.SelectedIndex);
            }
            else
            {
                  MessageBox.Show("No ha seleccionado ningún
apartado", "Mensaje", MessageBoxButtons.OK,
MessageBoxIcon.Information,
MessageBoxDefaultButton.Button1,
MessageBoxOptions.RightAlign);
            }
      }
      }
}
```

Código 3.1. Principal.cs

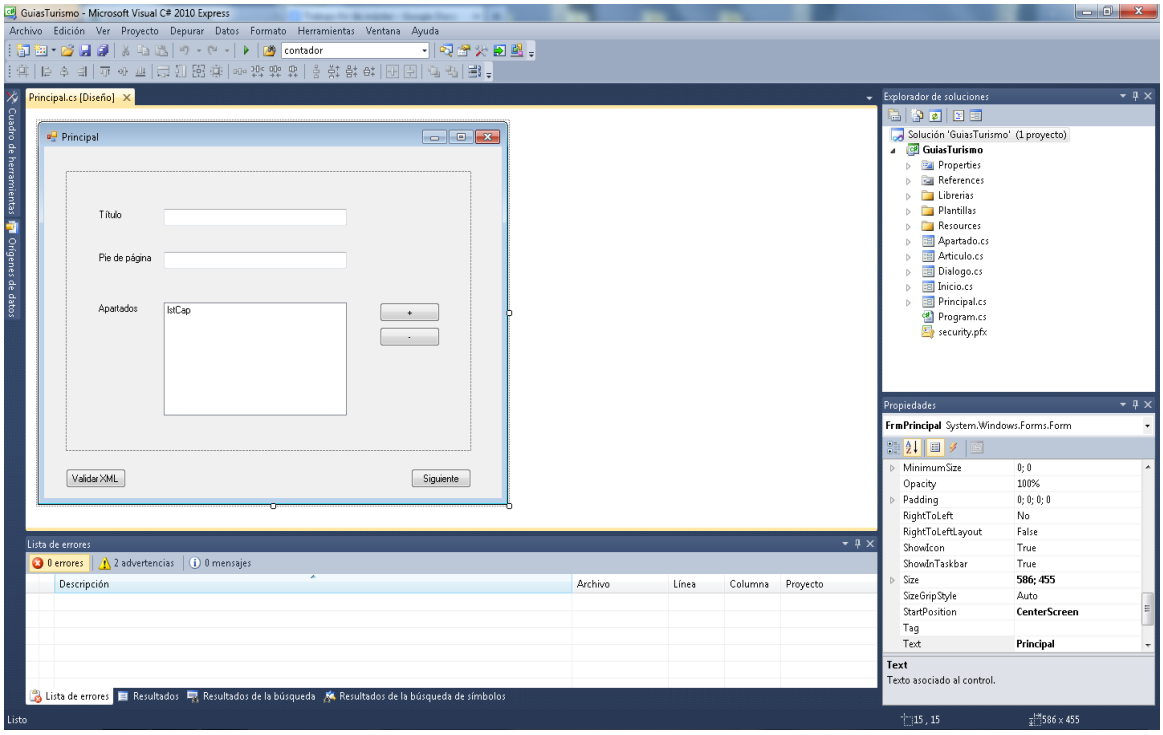


Figura 3.18. Entorno desarrollo

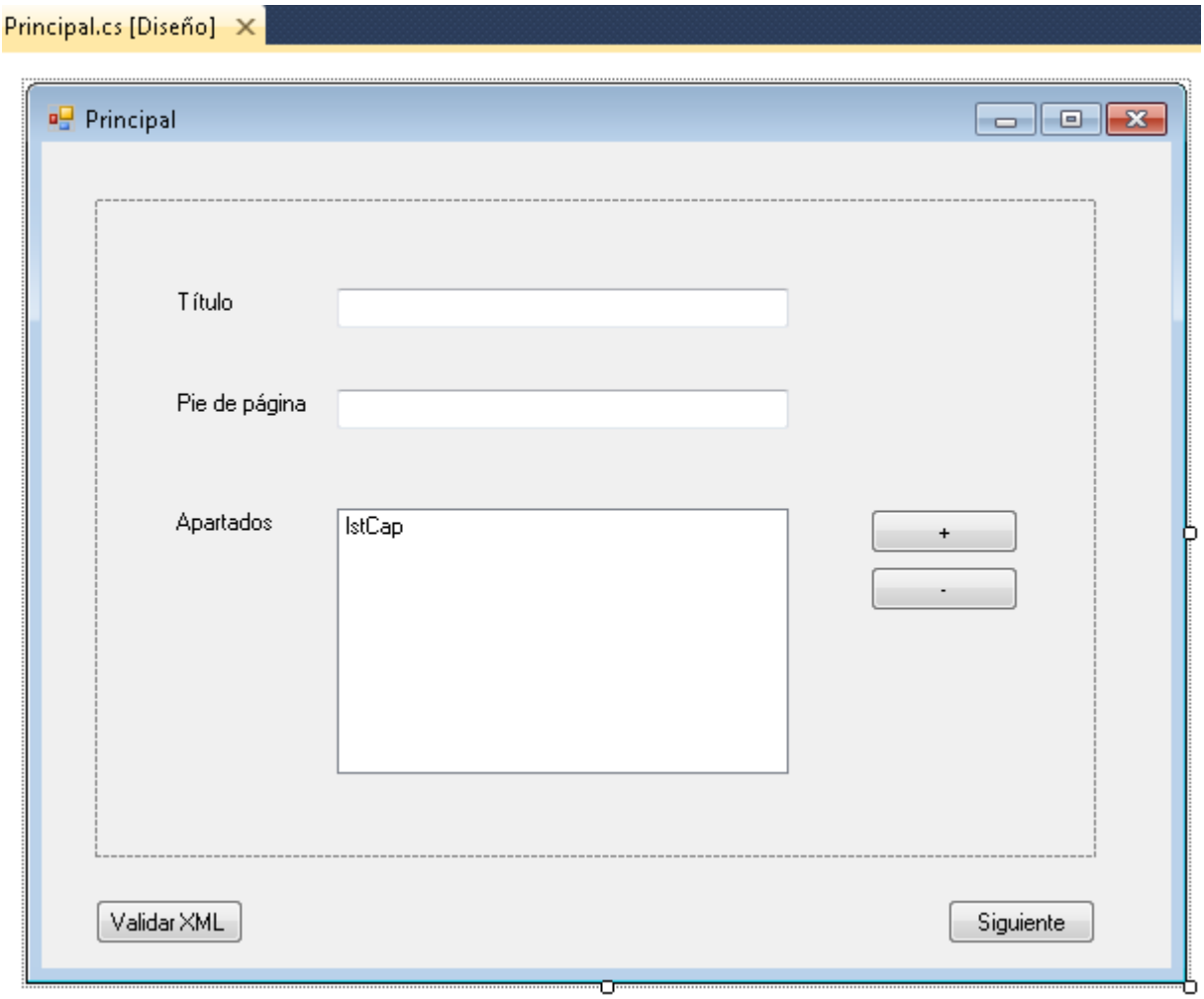


Figura 3.19. Formulario principal

Además de desarrollar la aplicación en C# hay que tener en cuenta el desarrollo de un archivo XML Schema y una plantilla XSLT [HOLZ03] [CRAN11].

El XML Schema se ha creado con el objetivo de validar que el XML generado por la aplicación sigue un esquema correcto con el fin de evitar errores, como se ha visto en este Máster las ventajas de uso de XML Schema con respecto a el uso de DTD (Document Type Definition) es que la sintaxis utilizada es XML y soporta la especificación de tipos de datos y tipos definidos por el usuario, así como comprobación de restricciones numéricas.

El XML Schema desarrollado para la aplicación es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
     <xs:element name="Webs">
          <xs:annotation>
               <xs:documentation>XML que permite
```

```
representar HTML5</xs:documentation>
          </xs:annotation>
          <xs:complexType>
               <xs:sequence minOccurs="1"
maxOccurs="unbounded">
                    <xs:element name="Web" type="tipoWeb"
minOccurs="1" maxOccurs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:complexType name="tipoWeb">
          <xs:sequence>
               <xs:element name="Enlaces">
                    <xs:complexType>
                         <xs:sequence>
                              <xs:element name="Enlace"
type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
                         </xs:sequence>
                    </xs:complexType>
               </xs:element>
               <xs:element name="Apartado" minOccurs="1"
maxOccurs="unbounded">
                    <xs:complexType>
                         <xs:sequence>
                              <xs:element
name="Introduccion" minOccurs="1" maxOccurs="1">
                                   <xs:complexType>
                                        <xs:sequence>
                                             <xs:element
name="Cabecera" type="xs:string"  minOccurs="1"
maxOccurs="1"/>
                                             <xs:element
name="Texto" type="xs:string"  minOccurs="1"
maxOccurs="1"/>
                                        </xs:sequence>
                                   </xs:complexType>
                              </xs:element>
                              <xs:element name="Articulos">
                                   <xs:complexType>
                                        <xs:sequence>
                                             <xs:element
name="Articulo" minOccurs="1" maxOccurs="unbounded">

     <xs:complexType>

     <xs:sequence>

     <xs:element name="Cabecera" type="xs:string"
minOccurs="1" maxOccurs="1"/>
```

```
     <xs:element name="Texto" type="xs:string"
minOccurs="1" maxOccurs="1"/>

     <xs:element name="Video" type="xs:string"
minOccurs="0" maxOccurs="1"/>

     <xs:element name="Imagen" type="xs:string"
minOccurs="0" maxOccurs="1"/>

     </xs:sequence>

     </xs:complexType>
                                                  </xs:element>
                                             </xs:sequence>
                                        </xs:complexType>
                                   </xs:element>
                              </xs:sequence>
                         </xs:complexType>
                    </xs:element>
                    <xs:element name="Pie" type="xs:string"
minOccurs="0" maxOccurs="1"/>
          </xs:sequence>
          <xs:attribute name="Titulo" use="required">
          </xs:attribute>
     </xs:complexType>
</xs:schema>
```

Código 3.2. XML Schema

Un ejemplo de plantilla XSLT para la aplicación es la siguiente:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:template match="/">
          <xsl:for-each select="Webs/Web">
               <html>
                    <head>
                         <title>
                              <xsl:value-of
select="@Titulo"/>
                         </title>
                         <link rel="stylesheet"
type="text/css" href="estilo.css"/>
                         <meta charset="utf-8"/>
                    </head>
                    <body>
```

```
                         <header>
                              <h1>
                                   <xsl:value-of
select="@Titulo"/>
                              </h1>
                         </header>
                         <nav>
                              <ul>
                                   <xsl:for-each
select="Enlaces/Enlace">
                                        <li>
                                             <a href="#">

     <xsl:value-of select="."/>
                                             </a>
                                        </li>
                                   </xsl:for-each>
                              </ul>
                         </nav>
                         <xsl:for-each select="Apartado">
                         <section>
                              <section class="intro">
                                   <header>
                                        <h2>
                                             <xsl:value-of
select="Introduccion/Cabecera"/>
                                        </h2>
                                        <xsl:value-of
select="Introduccion/Texto"/>
                                   </header>
                              </section>
                              <section>
                                   <xsl:for-each
select="Articulos/Articulo">
                                        <article>
                                             <header>
                                                  <h3>

     <xsl:value-of select="Cabecera"/>
                                                  </h3>
                                             </header>
                                             <xsl:value-of
select="Texto"/>
                                             <xsl:for-each
select="Imagen">
                                                  <br/>
                                                  <img>

     <xsl:attribute name="src"><xsl:value-of
select="."/></xsl:attribute>
```

```
                                                    </img>
                                                    <br/>
                                                </xsl:for-
each>
                                                <xsl:for-each
select="Video">
                                                    <br/>
                                                    <video
controls="controls">

     <source type="video/mp4">

     <xsl:attribute name="src"><xsl:value-of
select="."/></xsl:attribute>

     </source>
                                                                Your
browser does not support the video tag.
                                                    </video>
                                                    <br/>
                                                </xsl:for-
each>
                                            </article>
                                        </xsl:for-each>
                                    </section>
                                </section>
                                </xsl:for-each>
                                <footer>
                                    <xsl:value-of select="Pie"/>
                                </footer>
                            </body>
                        </html>
                    </xsl:for-each>
     </xsl:template>
</xsl:stylesheet>
```

Código 3.3. XSLT

La contrucción del XML en la aplicación se realiza entre otros mediante la librería System.Xml.Linq y System.Xml. *Language-Integrated Query* (LINQ) [LINQ] [KIMM08] es un conjunto de características presentadas en Visual Studio 2008 que añade capacidades de consulta a la sintaxis de los lenguajes C# y Visual Basic. Visual Studio incluye ensamblados de proveedores para LINQ que habilitan el uso de LINQ con colecciones de .NET Framework, bases de datos SQL Server, conjuntos de datos de ADO.NET y documentos XML.

La librería anteriormente indicada contiene las clases para LINQ to XML. LINQ to XML es una interfaz de programación XML en memoria que permite la creación y modificación de documentos XML.

Con LINQ to XML es posible:

- Cargar XML a partir de archivos o secuencias.
- Serializar XML a archivos o secuencias.
- Crear árboles XML desde cero usando la construcción funcional.
- Consultar árboles XML mediante consultas LINQ.
- Manipular árboles XML en memoria.
- Validar árboles XML mediante XSD.
- Usar una combinación de estas características para transformar las formas de los árboles XML.

De los elementos de la librería System.Xml.Linq que se han utilizado se pueden destacar los siguientes:

- XElement: elemento necesario para cargar y analizar un documento XML. Permite la creación y carga de un documento XML y su consulta. Posee un conjunto de propiedades que permiten navegar por el documento, como obtener el primer atributo de un elemento (FirstAttribute), obtener el primer nodo secundario de un nodo (FirstNode) o métodos para agregar el contenido especificado inmediatamente antes de este nodo (AddBeforeSelf) o después (AddAfterSelf), devolver una colección de los elementos antecesores de un nodo (Ancestors) o una colección de los atributos de un elemento (Attributes) entre otros. Pertenece a la librería System.Xml.Linq
- XmlSchemaSet: contiene una caché de esquemas del lenguaje de definición de esquemas XML (XSD). Esto nos permite asociar un esquema a un documento XML. El esquema se utiliza para validar la adecuación del documento XML al modelo creado. Se hace uso del evento ValidationEventHandler para controlar los errores y las advertencias de validación del esquema definidas. Pertenece a la librería System.Xml, que proporciona compatibilidad basada en estándares para procesar XML. En nuestro caso el más importante sería el de validación Esquemas XSD de la W3C. Pertenece a la librería System.Xml.Linq
- XDocument: permite representar un documento XML. Pertenece a la librería System.Xml.Linq
- XmlWriter: representa un sistema de escritura que constituye un medio no almacenado en caché y de sólo avance para generar secuencias o archivos con datos XML. Pertenece a la librería System.Xml
- XslCompiledTransform: es el procesador XSLT de Microsoft .NET Framework. Transforma datos XML utilizando una hoja de estilos XSLT. Pertenece a la librería System.Xml.
- XmlValidatingReader: representa un lector que proporciona validación de definiciones de tipos de documentos (DTD), de esquemas reducidos de datos XML (esquemas XDR) y del lenguaje de definición de esquemas XML (esquemas XSD). Es recomendable el uso de XmlSchemaSet en detrimento de XmlValidatingReader ya que esta última está en estado deprecated (obsoleta). Pertenece a la librería System.Xml.

Al finalizar la aplicación obtenemos como resultado una página HTML como se puede ver en la siguiente figura:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Titulo ejemplo</title>
    <link rel="stylesheet" type="text/css"
href="estilo.css" />
    <meta charset="utf-8" />
  </head>
  <body>
    <header>
      <h1>Titulo ejemplo</h1>
    </header>
    <nav>
      <ul>
        <li>
          <a href="#">Apartado 1</a>
        </li>
        <li>
          <a href="#">Apartado 2</a>
        </li>
      </ul>
    </nav>
    <section>
      <section class="intro">
        <header>
          <h2>Apartado 1</h2>Introducción apartado
1</header>
      </section>
      <section>
        <article>
          <header>
            <h3>Articulo 1</h3>
          </header>Texto árticulo 1
            <br/>
            <img src="../imagen1.jpg" />
            <br/>
          </article>
      </section>
    </section>
    <section>
      <section class="intro">
        <header>
          <h2>Apartado 2</h2>Introducción apartado
2</header>
      </section>
```

```
      <section>
        <article>
          <header>
            <h3>Articulo 3</h3>
          </header>
            Texto árticulo 2
          </article>
      </section>
    </section>
    <footer></footer>
  </body>
</html>
```

Código 3.4. Ejemplo generación HTML

3.5 PRUEBAS

Se ha comprobado el buen funcionamiento de la aplicación mediante pruebas funcionales. Para detectar posibles errores se han ejecutado de manera manual operaciones en la aplicación. Las pruebas funcionales se proporcionan en el CD-ROM adjunto al proyecto.

Además se ha hecho uso del programa FXCop, nombrado anteriormente, para evaluar la calidad de código de la aplicación [FXCOb]. En la siguiente figura se puede ver el nivel de errores de código que tiene la aplicación.

Level	Item	Fix Category	Certainty	Rule
⚠	WindowsFormsApplication1.FrmPrincipal.#Apartados	Breaking	75%	Collection properties should be read only
✖	WindowsFormsApplication1.FrmPrincipal.#lstCap	Breaking	90%	Do not declare visible instance fields
⚠	WindowsFormsApplication1.Apartado.#ListaApartados	Breaking	75%	Collection properties should be read only
⚠	WindowsFormsApplication1.Apartado.#ApartadosArticulos	Breaking	75%	Collection properties should be read only
⚠	WindowsFormsApplication1.Apartado.#ListaArticulos	Breaking	75%	Collection properties should be read only
✖	WindowsFormsApplication1.Apartado.#lstArticulos	Breaking	90%	Do not declare visible instance fields
⚠	WindowsFormsApplication1.Librerias.ApartadoObject.#Articulos	Breaking	75%	Collection properties should be read only

Figura 3.20. Errores FxCop

Los dos errores, traducidos al español, que se repiten y que han sido ignorados son:

- Las propiedades de la colección deben ser de solo lectura: dicha regla se ha ignorado ya que en nuestra aplicación necesitamos que ciertas colecciones de elementos puedan reemplazarse.
- No declarar campos de instancia visibles: como regla general los campos de una clase deben ser declarados “private” o “internal” y para utilizarlos se debe emplear propiedades “getters” para la obtención de datos y “setters”para la modificación. Esta regla ha sido ignorada ya que tanto la variable “lstCap” como “lstArticulos” son objetos “ListBox” del propio Windows Form y por lo tanto no formar parte de dicha regla.

PARTE IV

Conclusiones

Capítulo 4
Conclusiones

En este apartado se muestran las conclusiones extraídas a lo largo del desarrollo del proyecto, junto con algunos comentarios para posibles ampliaciones del sistema de información desarrollado.

4.1. VERIFICACIÓN, CONTRASTE Y EVALUACIÓN DE LOS OBJETIVOS

Uno de los objetivos de este proyecto ha sido estudiar los fundamentos de MDA, algo que he desempeñado y escrito en el capítulo titulado "Estado del arte". También, en el mismo apartado, se ha investigado sobre la creación de plantillas XSLT para usarlas como lenguaje de transformación y generación de código HTML 5.

Para ello se ha integrado estas plantillas en una aplicación sencilla que permite generar guías turísticas, aunque se puede ampliar el uso de la misma para otros propósitos. Este último objetivo se ha detallado durante el capítulo de implementación.

Por lo tanto, se puede afirmar que se han cumplidos los objetivos planteados al inicio de este proyecto.

4.2. SÍNTESIS DEL MODELO PROPUESTO

Presentar de modo resumido el modelo, arquitectura, metodología, etc. que se haya propuesto a lo largo del desarrollo de la investigación del trabajo fin de máster.

4.3. APORTACIONES ORIGINALES

El desarrollo de la investigación realizada ha permitido obtener una serie de aportaciones que son descritas a continuación:

- El desarrollo dirigido por modelos es una realidad y es posible usar esta arquitectura para afrontar diversos proyectos.
- Las transformaciones XSLT y el uso de lenguajes de marcador permiten desarrollar software de una manera ágil y de calidad.

4.4. LÍNEAS DE INVESTIGACIÓN FUTURAS

Seguramente se encuentren opciones y detalles que son mejorables o que pueden añadirse a la funcionalidad del sistema, pero esto se deja como posibles líneas futuras de mejora. Entre ellas podemos citar:

- La realización de una aplicación mucho más genérica que pudiera "entender"/"interpretar" el lenguaje de marcado (XML) definido y a partir del mismo generar la aplicación específica para luego sólo tener que incorporar la plantilla XSLT
- Aplicación que a partir de UML obtenga un documento de marcado XML que corresponda a XSLT

- Al igual que se ha realizado una aplicación de escritorio, realizar una aplicación Web o una aplicación para móviles. Se podría, desde un punto de vista más empresarial, crear una aplicación móvil en la que el usuario pueda crearse su propia guía de viaje. Respecto a la misma idea, cabría la posibilidad de crear la misma aplicación para redes sociales, por ejemplo, Facebook, en la que el usuario pudiese compartir su guía.
- Añadir nuevas estructuras - atributos a la plantilla XSLT que genera HTML5.
- Creación de una aplicación en la que los elementos son Drag and drop.
- Además de generar HTML, posibilidad de generar documentos de otro formato, como PDF.

BIBLIOGRAFÍA

[ALBA10]: **J. Albahari, B. Albahari**; *C# 4.0 in a Nutshell: The Definitive Reference*; O'Reilly Media, enero 2010; ISBN 0596800959.

[AMBL09]: **S.W. Ambler;** *The Agile Unified Process (AUP)*; Canada, 2009.

[BELL02]: **J. Bell, M. Reynolds, B. Johansen y otros;** *Developing C# Windows Software: A Windows Forms Tutorial (Programmer to Programmer);* WROX Press Ltd, Julio 2002; ISBN-10: 186100737X, ISBN-13: 978-1861007377.

[CABO09]: **J. Cabot**; *Relación entre MDA, MDD y MDE;* en Internet, 2009; http://modeling-languages.com/es/blog/content/relaci-n-entre-mdamdd-y-mde

[CRAN11]: **Crane Softwrights**; *Practical transformation using XSLT and XPATH (XSL Transformations and the XML Path Language);* Crane Softwrights, noviembre 2011; ISBN 978-1-894049-25-5.

[ECLI]: **Eclipse ATL Transformation**; sitio Web oficial: http://www.eclipse.org/m2m/atl/atlTransformations/

[FXCOa]: **FxCop**; sitio Web oficial: http://www.microsoft.com/download/en/details.aspx?id=6544

[FXCOb]: **Sociedad Informática del Gobierno Vasco***; FxCop. Manual de usuario v1.0*; EJIE S.A.

[GARC11]: **V. García Díaz;** "MDCD: Model-Driven Continuous Integration"; Universidad de Oviedo, 2011.

[GRAC10]: **J.P. Gracia Benítez;** "Marco para la transformación de modelos basado en gramáticas de atributos. Proyecto de fin de máster en sistemas inteligentes"; Universidad Complutense de Madrid, 2011.

[HOLZ03]: **S. Holzner;** *Sams Teach Yourself XML in 21 Days*; octubre 2003.

[HONG06]: **L. Hongming, Q. Lizhang, J. Xiaoping**; *Model transformation based on meta templates*; De Paul University, Chicago, 2006.

[HTML]: **HTML Specification**; sitio Web oficial: http://dev.w3.org/html5/spec/single-page.html

[KIMM08]: **P. Kimmel;** *LINQ Unleashed:for C#;* Sams Publishing, agosto 2008; ISBN-10: 0672329832, ISBN-13: 978-0672329838.

[KJAE09]: **M. Kjaer**; *HTML5 and CSS 3: The techniques you'll soon be using*; en Internet, 2009; http://net.tutsplus.com

[LINQ**]: LINQ (Language-Integrated Query);** sitio Web oficial: http://msdn.microsoft.com/es-es/library/bb397926.aspx

[LUDW10]: **Ludwig Maximilians Universität München**; *UWE – UML-based Web Engineering;* en Internet, 2010; http://uwe.pst.ifi.lmu.de/index.html

[MILL03a]: **J. Miller y J. Mukerji**; *MDA Guide Version 1.0.1*; OMG, 2003.

[MILL03b]: **J. Miller**; *Presentation of MDA*; en Internet, 2003; http://www.joaquin.net/MDA/

[MONT11]: **C.E. Montenegro Marín**; "Modelado específico de dominio para la construcción de learning objects independientes de la plataforma"; Universidad de Oviedo, 2011.

[OMG]: **OMG**; sitio Web oficial: http://www.omg.org

[OMG11]: **OMG**; *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version 1.1;* OMG, enero 2011.

[PELA07]: **B.C. Pelayo García-Bustelo**; "TALISMAN: Desarrollo ágil de Software con Arquitectras Dirigidas por Modelos"; Universidad de Oviedo, 2007.

[PEÑA07]: **J.C.Peña***;* "TicXLM: Generando diferentes interfaces de usuario a partir de una única especificación declarativa"; Universidad de Castilla-La Mancha, 2007.

[SCHM06]: **D. Schmidt**; "Model-Driven Engineering"; Vanderbilt University, 2006.

[UMLE]: **UMLet**; sitio Web oficial: www.umlet.com

[VALL03]: **A. Vallecillo**; "Model Driven Development PhD thesis"; Universidad de Málaga, 2003.

[VALL08]: **A. Vallecillo**; *Modelado de apliaciones software mediante MDA*; en Internet, 2008; http://www.lcc.uma.es/~canal/sabc/

[XML**]: XML Specification**; sitio Web oficial: http://www.w3.org/XML/

[XMLSa**]: XMLSpy** ; sitio Web oficial: http://www.altova.com/xmlspy.html

[XMLSb]: **Sociedad Informática del Gobierno Vasco***; XMLSpy. Manual de usuario v1.0*; EJIE S.A., 2007.

[XSLT]: **XSLT Specification**; sitio Web oficial: http://www.w3.org/TR/xslt20/

PARTE V

Anexos

Anexo A
Artículo en castellano

En este apartado se adjunta el artículo de divulgación en castellano del trabajo de fin de máster.

Generación de guías turísticas en HTML5 mediante un desarrollo dirigido por modelos

Francisco Núñez Sánchez, Cristina Pelayo García-Bustelo

Universidad Internacional de La Rioja
Avenida Gran Vía, 41 26002 Logroño
http://www.unir.net

Resumen: El objetivo principal ha sido estudiar los fundamentos del desarrollo dirigido por modelos para luego implementar una aplicación. Para ello se han investigado los principios y fundamentos del desarrollo dirigido por modelos y se ha enfocado al uso de XSLT como lenguaje de transformación. Se ha implementado una aplicación de creación de guías turísticas que genera código HTML5. Para el desarrollo se ha utilizado el lenguaje de programación C#, XML, XML Schema y XSLT.

Palabras clave: MDA, XSLT, guías turísticas, HTML5

1. INTRODUCCIÓN

Los objetivos de este proyecto son los siguientes:

- Estudiar cuáles son los principios y fundamentos de MDA.
- Investigar sobre la creación de plantillas XSLT para la generación de código HTML5.
- Integrar la/s plantilla/s en una aplicación sencilla para usuario sin conocimientos de programación.
- Creación de guía turística haciendo uso de las plantillas creadas.

Las exigencias del usuario van aumentando con el tiempo debido a las nuevas tecnologías. Actualmente en el mercado existen multitud de lenguajes de programación y una gran variedad de utilidades y *frameworks* que ayudan al programador a que sus tareas sean más fáciles tras su previo aprendizaje. Para intentar enfrentarnos a este problema tecnológico surge el desarrollo de software dirigido por modelos, que pretende usar los modelos, usados por ahora únicamente como documentación, como las entidades principales del desarrollo de software.

2. ESTADO DEL ARTE

2.1 El uso de MDA. Ingeniería dirigida por modelos

La ingeniería dirigida por modelos (siglas en inglés MDE) es una metodología de desarrollo de software que se basa en la creación de modelos o abstracciones [1]. Tiene como objetivo el aumentar la productividad mediante la compatibilidad entre sistemas, simplificando el proceso de diseño.

Un paradigma de modelado para el MDE se considera eficaz si los modelos son entendidos desde el punto de vista del usuario y pueden servir como base para la implementación de sistemas. Los modelos son desarrollados mediante la comunicación entre los gerentes de producto, diseñadores y miembros del equipo de desarrollo.

Los modelos se construyen para un determinado nivel de detalle o se construyen modelos completos, incluidas las acciones ejecutables. El código puede ser generado a partir de los modelos.

De acuerdo con Douglas C. Schmidt [4], las tecnologías MDE ofrecen un enfoque prometedor para hacer frente a la incapacidad de los lenguajes de tercera generación para aliviar la complejidad de las plataformas de desarrollo y expresar conceptos de dominio de una manera eficaz.

A partir de este concepto surge también el concepto de *model-driven development* (MDD), es decir, desarrollo dirigido por modelos [5]. MDD es un paradigma de desarrollo que considera los modelos de software como principal elemento del proceso de desarrollo. Normalmente, además, a partir de estos modelos se genera de una forma semi-automática el código.

La iniciativa MDE más conocida es MDA, ofrecida por el *Object Management Group* (OMG)[2] [3].

2.2 Arquitectura dirigida por modelos

MDA responde a las siglas de *"Model Driven Architecture"*, que significa "Arquitectura dirigida por modelos". Esta arquitectura fue propuesta por la OMG (*Object Management Group*), consorcio de industria informática sin ánimo de lucro, que tiene como uno de sus principales objetivos el desarrollo de estándares.

El OMG es una organización de compañías de sistemas de información creada en 1990 con el fin de potenciar el desarrollo de aplicaciones orientadas a objetos distribuidas. Esta organización ha definido estándares importantes como UML, CORBA, MOF, entre otros.

En los últimos años, el modelado en el desarrollo de cualquier tipo de software ha tomado mayor interés e importancia, debido a la facilidad que ofrece un buen diseño tanto a la hora de desarrollar como al hacer la integración y mantenimiento de sistemas de software.

Es así que en el año 2001, el OMG definió un marco de trabajo nuevo llamado MDA. La clave del MDA es la importancia de los modelos en el proceso de desarrollo de software. MDA propone la definición y uso de modelos a diferente nivel de abstracción, así como la posibilidad de la generación automática de código a partir de los modelos definidos y de las reglas de transformación entre dichos modelos.

MDE se puede aplicar a la ingeniería del software, sistemas, y datos. La primera herramienta de apoyo a MDE fue CASE (*Computer-Aided Software Engineering*) desarrollada en los ochenta. Con algunas variaciones de las definiciones de modelado se creó lo que se conoce en la actualidad por Lenguaje Unificado de Modelado (UML).

MDA es una plataforma para desarrollo de aplicaciones, cuyo objetivo es aumentar la calidad y velocidad de desarrollo de aplicaciones, llevándolo a cabo mediante el aumento de nivel de abstracción, junto al uso de técnicas de modelado, de transformación del modelo y de generación de código.

MDA defiende la separación de la especificación de la funcionalidad de un sistema y su implementación, independientemente de la plataforma que se utilice.

Por lo tanto, MDA, basado en estándares de la OMG, separa la lógica de negocio y la de la plataforma. A su vez se basa en los principios de abstracción, automatización y estandarización.

La mayoría de las aplicaciones actuales se basan en una arquitectura básica de tres capas (three-tier), que son:

- Capa de presentación: la presentación al usuario, con las entradas de datos y las pantallas de consulta.
- Capa de lógica de negocio: donde se procesa la información.
- Capa de datos: el control del almacén de datos.

La arquitectura MDA se centra en el modelo y persigue tres objetivos básicos, todos ellos a través de la separación arquitectónica:

- La portabilidad: la característica que posee un software para ejecutarse en diferentes plataformas. El código fuente del software es capaz de reutilizarse en vez de crearse un nuevo código cuando el software pasa de una plataforma a otra. A mayor portabilidad menor es la dependencia del software con respecto a la plataforma.
- La interoperabilidad: condición mediante la cual sistemas heterogéneos pueden intercambiar procesos o datos. En otras palabras, habilidad que tiene un sistema o producto para trabajar con otros sistemas o productos sin un esfuerzo especial por parte del cliente.

- La reusabilidad: hace referencia a la capacidad del software, en parte o completamente, para ser usado en otro proyecto.

La OMG, con esta definición de MDA, dispone de herramientas que permiten (véase la figura Figura 1):

- Especificar sistemas independientes de las plataformas que los soportan, modelando los mismos mediante modelos independientes de plataforma (PIMs, *Platform Independent Models*).
- Escoger la plataforma más adecuada para cada tipo de sistema y especificarla mediante modelos específicos de plataformas (PSMs, *Platform Specific Models*).
- Transformar las especificaciones de los sistemas (los PIMs) a las especificaciones de las plataformas (los PSMs).

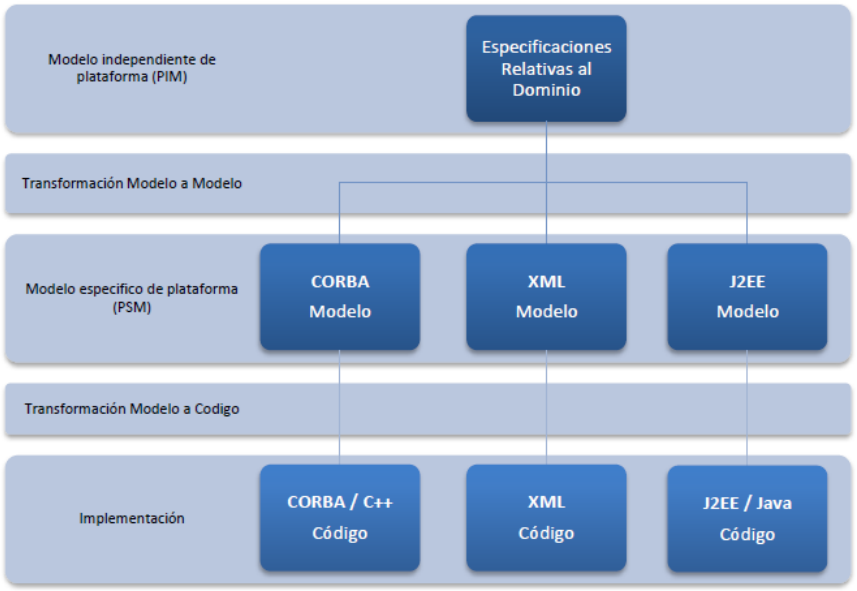


Figura 1: MDA

La transformación de modelos es fundamental en los diversos enfoques del desarrollo de software dirigido por modelos. Esta tecnología permite caracterizar artefactos capaces de traducir automáticamente modelos que se ajustan a un determinado metamodelo origen, en modelos que se ajustan a un determinado metamodelo objetivo (transformaciones modelo a modelo). Asimismo, estas tecnologías permiten también realizar transformaciones entre modelos y otros formatos de representación (p.ej., entre modelos y formatos textuales, dando lugar a las denominadas transformaciones de modelo a texto y de texto a modelo).

En los últimos años se han definido una gran cantidad de lenguajes de transformación, cada uno con diferentes características. Existen distintas propuestas como lenguajes de transformación (XSLT, ATL, ETL, QVT, etc.). En este proyecto la idea es la de utilizar el lenguaje de transformación XSLT.

2.3 XSLT

XSLT es un lenguaje estándar de la World Wide Web Consortium (W3C) [7] y cuenta con un extenso soporte tecnológico. Las hojas de estilo XSLT, aunque el término de hojas de estilo no se aplica sobre la función directa del XSLT, realizan la transformación del documento utilizando una o varias reglas de plantilla. Estas reglas de plantilla, unidas al documento fuente a transformar, alimentan un procesador de XSLT, el cual realiza las transformaciones deseadas poniendo el resultado en un archivo de salida, o, como en el caso de una página web, las hace directamente en un dispositivo de presentación tal como el monitor del usuario.

XSLT permite transformar un documento XML en otro formato. Este formato puede ser HTML, XML o cualquier otro. En consecuencia, XSLT podría ser utilizado para transformar los modelos, que están representados en el formalismo de XMI [6].

Como principales características, podemos decir que XSLT tiene una naturaleza híbrida, pues permite tanto construcciones imperativas como declarativas. En su parte declarativa, la aplicación del pattern-matching se hace de forma ordenada y recursiva, por lo que XSLT es determinista. Además, desde la perspectiva de metamodelado, efectúa la transformación de grafos, que son árboles, como representación de modelos, y utiliza XPath como lenguaje de consulta. Actualmente, XSLT es muy utilizado ya que permite separar contenido y presentación, aumentando así la productividad.

El lenguaje XSLT tiene sus propias etiquetas, las hay que se utilizan para definir plantillas y hacer llamadas a las mismas, otras para emular estructuras condicionales, de selección, etc. También para funciones como aplicar atributos del fichero XML original. De esta manera, se puede combinar la utilización del archivo origen en UsiXML con un archivo de tipo XSL que haga las veces de plantilla de transformación, para obtener el deseado fichero de la implementación final. De las transformaciones, se pueden obtener dos tipos de ficheros: texto y xml (etiquetado). La forma de realizar transformaciones será totalmente diferente, ya que al estar XSLT basado en XML el texto ha de introducirse mediante etiquetas, es decir, habrá que indicar expresamente que se desea introducir texto. En caso de tener como resultado un archivo etiquetado, se podrán incluir las etiquetas de forma normal.

A continuación se puede observar la estructura de un documento XSLT, donde se puede observar la plantilla raíz, que es la primera que se ejecutará y que a su vez puede ir aplicando las plantillas que se deseen o aplicando cualquier regla.

Ejemplo de estructura XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0">
 <xsl:template match='/'>

 </xsl:template>
</xsl:stylesheet>
```

Relativo a XSLT podemos también resaltar la importancia de XML Schema como lenguaje para definición de *schemas* en el que se expresa el modelo a seguir en las instancias (se describe lo único válido en ellas) utilizando una estructura en árbol con un detallado sistema de tipos y propiedades de orientación a objetos o XSL-FO como lenguaje especializado en la transformación de un documento XML con el formato de presentación final.

XML Schema facilita la descripción de un modelo de dominio gracias a las siguientes características: propiedades de orientación a objetos (herencia, extensibilidad, polimorfismo), capacidad de instanciación, espacios de nombre, riqueza de tipos de datos predefinidos y posibilidades de personalización, expresión formal de relaciones entre elementos, estructura jerárquica en forma de árbol, atributos para recoger las propiedades de los elementos, validación formal de las instancias mediante un parser genérico y transformaciones XSLT para exportar o importar información entre modelos.

2.4 XML

XML, siglas en inglés de Extensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes especficos (de la misma manera que HTML es a su vez un lenguaje defnido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su defiición son XHTML, SVG, MathML.

XML se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Su uso está en bases de datos, editores de texto, hojas de cálculo, etc. XML es una tecnología sencilla que se integra y complementa con otras. Actualmente tiene un papel importante debido a que permite la compatibilidad entre sistemas para compartir información de una manera segura, fiable y fácil.

Sus principales ventajas son:

- Es extensible: después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan bugs y se acelera el desarrollo de aplicaciones.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.

XML es un meta-lenguaje que permite definir formalmente lenguajes de marcas, de forma que el lenguaje definido sirva para expresar un modelo concreto de información. Se puede sintetizar en 3 niveles la arquitectura de abstracción XML (ver Figura 2):

- Lenguaje genérico de XML Schema. Constituye un meta-modelo de datos. Permite describir descripciones de datos. Define el meta-lenguaje que puede emplearse para construir lenguajes específicos.
- Lenguaje particular definido en un Schema XML. Constituye un modelo de datos. Permite describir datos. Define el lenguaje que se empleará para describir ciertos datos particulares.
- Documento XML. Descripción de datos en base al lenguaje definido por el schema.

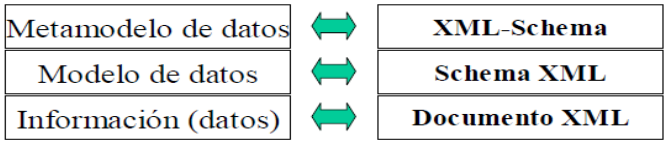


Figura 2: Transformación XML

2.5 HTML5

HTML5 (*HyperText Markup Language*, versión 5) es la quinta revisión importante del lenguaje básico de la World Wide Web [8], HTML. HTML5 especifica dos variantes de sintaxis para HTML: HTML (text/html), la variante conocida como HTML5 y una variante XHTML conocida como sintaxis XHTML5 que deberá ser servida como XML (XHTML) (application/xhtml+xml). HTML y XHTML por primera vez se han desarrollado en paralelo.

Todavía se encuentra en modo experimental, lo cual indica la misma W3C; aunque ya es usado por múltiples desarrolladores web por sus avances, mejoras y ventajas.

Al no ser reconocido en versiones más antiguas de navegadores por sus nuevas etiquetas, se le recomienda al usuario actualizar a la versión más actual. El desarrollo de este código es regulado por el Consorcio W3C.

HTML5 añade etiquetas para manejar la Web Semántica (Web 3.0): header, footer, article, nav, time (fecha del contenido), link rel=‘’ (tipo de contenido que se enlaza) que permiten describir cual es el significado del contenido. Por ejemplo su importancia, su finalidad y las relaciones que existen. No tienen especial impacto en la visualización, se orientan a buscadores. Los buscadores podrán indexar e interpretar esta meta información para no buscar simplemente apariciones de palabras en el texto de la página. Permite incorporar a las páginas ficheros RDF / OWL (con meta información) para describir relaciones entre los términos utilizados.

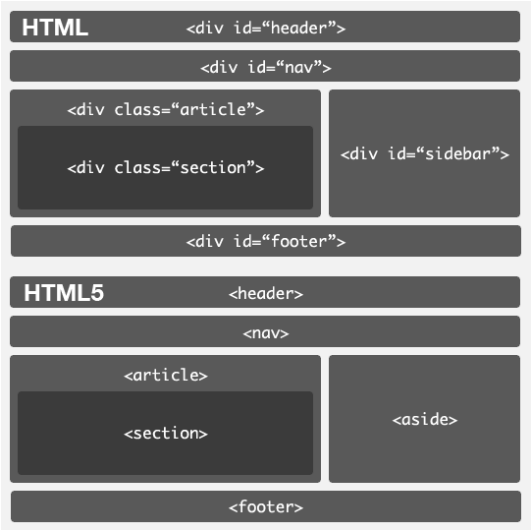


Figura 3: Comparativa estructura HTML y HTML5

HTML5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web actuales[9]. Algunos de estos cambios son técnicamente similares a las etiquetas y , pero tienen un significado semántico, como por ejemplo <nav>(bloque de navegación del sitio web) y <footer>. Otros elementos proporcionan nuevas funcionalidades a través de una interfaz estandarizada, como los elementos <audio>y <video>. Ver figura 3.

Existen mejoras en el elemento <canvas>, capaz de renderizar en algunos navegadores elementos 3D. Algunos elementos de HTML 4.01 han quedado obsoletos, incluyendo elementos puramente de presentación, como <font>y , cuyos

efectos son manejados por las hojas de estilo (CSS). También se resalta la importancia del scripting DOM para el comportamiento de la web.

Cuando los navegadores renderizan una página, construyen un "Modelo de Objeto de Documento" (*Document Object Model* - DOM), una colección de objetos que representan los elementos del HTML en la página. Cada elemento (<p>, , , etc.) es representado en el DOM por un objeto diferente.

Todos los objetos DOM comparten unas características comunes, aunque algunos poseen más que otros. En los navegadores que soportan rasgos del HTML5, algunos objetos tienen una única propiedad y observando al DOM se puede saber las características que soporta el navegador.

3. IMPLEMENTACIÓN

Las piezas fundamentales para el desarrollo de aplicaciones de escritorio haciendo uso de Visual C# son los formularios. Un formulario es, en última instancia, una hoja en blanco que el desarrollador rellena con controles, para crear una interfaz de usuario, y con código, para procesar los datos. Los controles son objetos contenidos en objetos de formulario. Cada tipo de control tiene su propio conjunto de propiedades, métodos y eventos que lo hacen adecuado para un propósito en particular. Es posible manipular los controles y escribir código para agregar controles dinámicamente, en tiempo de ejecución.

Además de desarrollar la aplicación en C# hay que tener en cuenta el desarrollo de un archivo XML Schema y una plantilla XSLT.

El XML Schema se ha creado con el objetivo de validar que el XML generado por la aplicación sigue un esquema correcto con el fin de evitar errores. Como se ha visto en este Máster, las ventajas de uso de XML Schema con respecto al uso de DTD (*Document Type Definition*) es que la sintaxis utilizada es XML y soporta la especificación de tipos de datos y tipos definidos por el usuario, así como comprobación de restricciones numéricas.

La contrucción del XML en la aplicación se realiza entre otros mediante la librería System.Xml.Linq y System.Xml. *Language-Integrated Query* (LINQ) es un conjunto de características presentadas en Visual Studio 2008 que añade capacidades de consulta a la sintaxis de los lenguajes C# y Visual Basic. Visual Studio incluye ensamblados de proveedores para LINQ que habilitan el uso de LINQ con colecciones de .NET Framework, bases de datos SQL Server, conjuntos de datos de ADO.NET y documentos XML.

La librería anteriormente indicada contiene las clases para LINQ to XML. LINQ to XML es una interfaz de programación XML en memoria que permite la creación y modificación de documentos XML.

Con LINQ to XML es posible:

- Cargar XML a partir de archivos o secuencias.
- Serializar XML a archivos o secuencias.
- Crear árboles XML desde cero usando la construcción funcional.
- Consultar árboles XML mediante consultas LINQ.
- Manipular árboles XML en memoria.
- Validar árboles XML mediante XSD.
- Usar una combinación de estas características para transformar las formas de los árboles XML.

Se ha hecho uso del programa FXCop para evaluar la calidad de código de la aplicación.

4. CONCLUSIONES

Uno de los objetivos de este proyecto ha sido estudiar los fundamentos de MDA, algo que he desempeñado y escrito en el cápitulo titulado “Estado del arte”. También, en el mismo apartado, se ha investigado sobre la creación de plantillas XSLT para usarlas como lenguaje de transformación y generación de código HTML 5.

Para ello se ha integrado estas plantillas en una aplicación sencilla que permite generar guías turísticas, aunque se puede ampliar el uso de la misma para otros propósitos. Este último objetivo se ha detallado durante el capítulo de implementación.

El desarrollo de la investigación realizada ha permitido obtener una serie de aportaciones que son descritas a continuación:

- El desarrollo dirigido por modelos es una realidad y es posible usar esta arquitectura para afrontar diversos proyectos.
- Las transformaciones XSLT y el uso de lenguajes de marcador permiten desarrollar software de una manera ágil y de calidad.

5. LÍNEAS FUTURAS

Seguramente se encuentren opciones y detalles que son mejorables o que pueden añadirse a la funcionalidad del sistema, pero esto se deja como posibles líneas futuras de mejora. Entre ellas podemos citar:

- La realización de una aplicación mucho más genérica que pudiera “entender”/”interpretar” el lenguaje de marcado (XML) definido y a partir del mismo generar la aplicación específica para luego sólo tener que incorporar la plantilla XSLT.
- Aplicación que a partir de UML obtenga un documento de marcado XML que corresponda a XSLT.
- Al igual que se ha realizado una aplicación de escritorio, realizar una aplicación Web o una aplicación para móviles. Se podría, desde un punto de vista más empresarial, crear una aplicación móvil en la que el usuario pueda crearse su propia guía de viaje. Respecto a la misma idea, cabría la posibilidad de crear la misma aplicación para redes sociales, por ejemplo, Facebook, en la que el usuario pudiese compartir su guía.
- Añadir nuevas estructuras - atributos a la plantilla XSLT que genera HTML5.
- Creación de una aplicación en la que los elementos son Drag and drop.
- Además de generar HTML, posibilidad de generar documentos de otro formato como PDF.

6. REFERENCIAS

[1] *HONGMING, Liu and LIZHANG, Qin and XIAOPING, Jia*, Model transformation based on meta templates.
International Conference on Software Engineering Research & Practice. 2006.
De Paul University. School of Computer Science, Telecommunications and Information Systems.
[2] *CABOT, Jordi*, Relación entre MDA, MDD y MDE
http://modeling-languages.com/es/blog/content/relaci-n-entre-mdamdd-y-mde. 2009.

[3] OMG, http://www.omg.org
[4] *SCHMIDT, Douglas*, Model-Driven Engineering
Vanderbilt University. 2006.
[5] *VALLECILLO, Antonio*, Model Driven Development
Universidad de Málaga. 2003.
[6] *PEÑA, Juan Carlos*, TicXML: Generando diferentes interfaces de usuario finales a partir de una única especificación declarativa
Universidad de Castilla-La Mancha. 2007.
[7] XSLT Specification, http://www.w3.org/TR/xslt20/
[8] HTML5 Specification, http://dev.w3.org/html5/spec/single-page.html
[9] *KJAER, Mads*, HTML 5 and CSS 3: The Techniques You'll Soon Be Using, http://net.tutsplus.com, 2009

Anexo B
Artículo en inglés

En este apartado se adjunta el artículo de divulgación en inglés del trabajo de fin de máster.

Generation of tourist guides in HTML5 through a model-driven development

Francisco Nuñez Sanchez, Cristina Pelayo Garcia-Bustelo

Universidad Internacional de La Rioja
Avenida Gran Vía, 41 26002 Logroño
http://www.unir.net

Abstract: The main purpose was to study the fundamentals of model-driven development and then implement an application. Therefore, we investigated the principles and basis of model-driven development and has focused on the use of XSLT transformation language. We have implemented an authoring application that generates code guidebooks HTML5. For development we used the programming language C#, XML, XML Schema and XSLT.

Keywords: MDA, XSLT, guidebooks, HTML5

1. INTRODUCTION

The purposes of this project are:

- Study which are the principles and basis of MDA.
- Investigate the creation of XSLT templates for code generation HTML5.
- Finally, integration of the templates in a simple application for users without programming skills.
- Creating guidebook using templates created before.

User requests are increasing due to new technologies. Currently, there are many programming languages and a variety of utilities and frameworks on the market to facilitate programmers their tasks after learning them. To face up this problem, driven software development models emerge, which intends to use the models used so far only as documentation, such as the principals of software development.

2. STATE OF THE ART

2.1 The use of MDA. Model Driven Engineering

Model-driven engineering (MDE) is a software development methodology based on modeling or abstractions [1]. The MDE approach is meant to increase productivity by maximizing compatibility between systems, simplifying the process of design.

A modeling paradigm for MDE is considered effective if its models make sense from the point of view of a user that is familiar with the domain, and if they can serve as a basis for implementing systems. The models are developed through extensive communication among product managers, designers, developers and users of the application domain.

The models are constructed to a certain level of detail or complete models are built including executable actions. Code can be generated from the models.

According to Douglas C. Schmidt [4], model-driven engineering technologies offer a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively.

From this concept comes also the concept of model-driven development(MDD) [5]. MDD is a development paradigm that considers the models as the main software development process. Furthermore, from these models, the code is generated semi-automatically.

The best known MDE initiative is MDA, offered by the Object Management Group(OMG) [2] [3].

2.2 Model Driven Architecture

Model Driven Architecture (MDA) is an architecture that was proposed by the OMG (Object Management Group), computer industry consortium of nonprofit, which has as one of its main objectives the development of standards. OMG is an organization of information systems companies founded in 1990 originally aimed at setting standards for distributed object-oriented systems. This organization has set important standards such as UML, CORBA, MOF, among others.

In recent years, modeling in the development of any type of software has become more interesting and important, because of the ease that good design offers both in developing and making the integration and maintenance of software systems.

In 2001, the OMG has defined a new framework called MDA. MDA key is the importance of models in the software development process. MDA proposes the definition and use of models at different levels of abstraction, and the possibility of automatic code generation from models defined and the rules of transformation between these models.

MDE can be applied to software engineering, systems and data. The first tool to support MDE was CASE (Computer-Aided Software Engineering) developed in the eighties. With some changes in definitions of modeling is created what is known today for Unified Modeling Language (UML).

MDA is a platform for application development. It aims to increase the quality and speed of application development, holding it by increasing level of abstraction and the use of modeling techniques, model transformation and code generation.

One of the main aims of the MDA is to separate the specification of the functionality of a system and its implementation, regardless of platform used.

Therefore, MDA, based on the OMG standards, separates the business logic and platform. It is also based on the principles of abstraction, automation and standardization.

Most current applications are based on a basic architecture of three-tier, which are:

- Presentation tier: the presentation to the user, data entry and query screens.
- Application tier: information processing.
- Data tier: information is stored and retrieved.

The MDA architecture focuses on the model and has three basic objectives, all of them by three-tier architecture:

- Portability: the software has a feature to run on different platforms. The software source code can be reused instead of creating a new code when the software moves from one platform to another. The more portability, less dependence of the software relative to the platform.
- Interoperability: ability of a system or product to work with other systems or products without special effort by the customer.
- Reusability: ability of software to be used in another project.

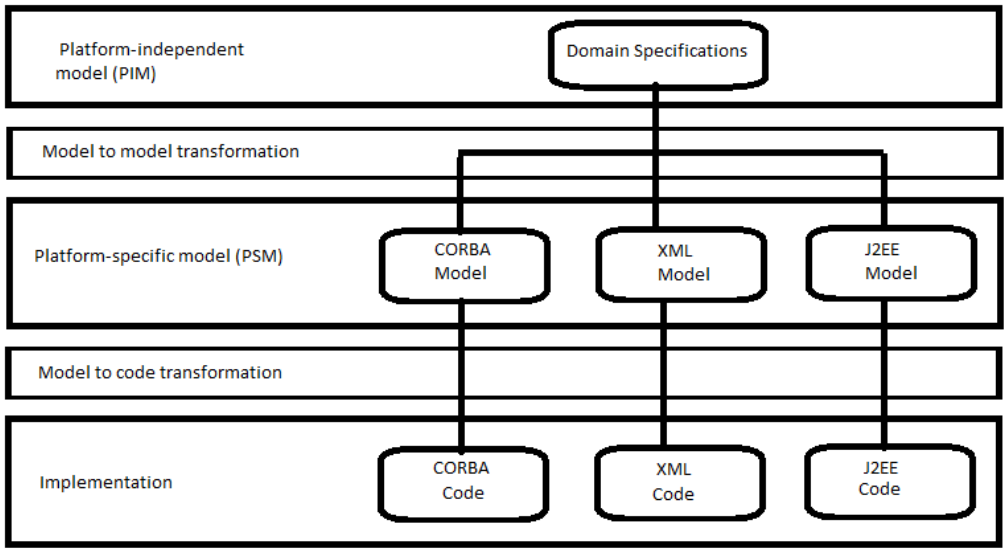


Figure 1: MDA

MDA has tools that allow (see figureFigura 1):

- Specify platform independent systems that support them, modeling them using platform independent models (PIMs, Platform Independent Models).
- Choosing the most appropriate platform for each system type and specify it using platform specific models (PSMs, Platform Specific Models).
- Transforming specifications of the systems (PIMs) to the specifications of the platforms (PSMs).

Transformation of models is very important in many approaches to software development model driven. This technology can characterize devices that can automatically translate models to set a particular source metamodel, in models that set a particular target metamodel (model to model transformations). In addition, these technologies also allow transformations between models and other forms of representation (for example, between models and textual formats, leading to so-called model transformations to text and text-to-model).

In recent years, a large amount of processing languages have been defined, each one with different characteristics. There are many proposals for transformation languages (XSLT, ATL, ETL, QVT...). In this project the idea is to use the XSLT transformation language.

2.3 XSLT

XSLT is a standard language of the World Wide Web Consortium (W3C) [7] and has an important technological support. XSLT style sheets, although the term stylesheet is not applied to the direct role of XSLT, perform the transformation of the document using one or more template rules. These template rules attached to the document feed source to transform XSLT processor, which performs the desired transformations putting the result in an output file, or, as in the case of a web page, makes directly to a display device as the user's monitor.

XSLT can transform an XML document into another format. This format can be HTML, XML or otherwise. Thus, XSLT could be used to transform the models, which are represented in the formalism of XMI [6].

XSLT has a hybrid nature, because it allows both declarative and imperative constructions. In its declarative part, the application of pattern-matching is an orderly and recursive, so XSLT is deterministic. Furthermore, from the perspective of metamodeling, it performs the transformation of graphs, which are trees, as a representation of models, and uses XPath as query language. Currently, XSLT is widely used because it can separate the content and presentation, increasing productivity.

XSLT language has its own labels. Some are used to define templates and make calls to them, others to emulate conditional structures, selection, etc. They are also used for functions such as applying the original XML file attributes. This way, you can combine the use of the source file in UsiXML with a file of type XSL, which works as a template for transformation, to get the final implementation file. You can get two kind of files from transformations: text and xml (label). The way of performing transformations is different. This is because XSLT is XML based, so the text must be entered using labels. If you get a file labeled, the labels can be included normally. The following shows the structure of an XSLT document, where you can see the template root, which is the first to be executed and applying the desired templates or applying any rule.

XSLT Structure example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0">
 <xsl:template match='/'>
 в Ђ¦
 </xsl:template>
</xsl:stylesheet>
```

Talking about XSLT, we also highlight the importance of XML Schema as a language for defining schemas which expresses the model in the instances (described as valid only in them) using a tree structure with a detailed type system and object-oriented properties or XSL-FO as a specialized language in transforming an XML document with the final presentation format.

XML Schema provides a description of a domain model thanks to the following features: object-oriented properties (inheritance, extensibility, polymorphism), ability to instantiation, namespaces, a wealth of predefined data types and customization, formal expression relationships between elements, hierarchical structure as a tree, attributes to collect the properties of the elements, formal validation of instances with generic parser and XSLT transformations to export or import data between models.

2.4 XML

XML (Extensible Markup Language) is a markup language, developed by the World Wide Web Consortium (W3C). It is a simplification and adaptation of SGML and allows to define the grammar of specific languages (in the same way that HTML is a language defined by SGML). So XML is not really a particular language, but a way of defining languages for different needs. Some of these languages that use XML for its definition are XHTML, SVG, MathML, etc.

XML is proposed as a standard for exchanging structured information between different platforms. It is used in databases, text editors, spreadsheets, etc.. XML is a

simple technology that integrates and complements other. Currently has an important role because it allows compatibility between systems to share information in a safe, reliable and easy way.

XML main advantages are:

- It is extensible: After designed and put into production, it is possible to extend XML with the addition of new label, so you can continue without complications.
- The parser is a standard component, so it is not necessary to create a specific one to each version of XML. This enables the use of any of the available parser. This will prevent bugs and accelerates application development.
- If another user wants to use a document created in XML, it is easy to understand its structure and process. Improves compatibility between applications.

XML is a markup language for defining markup languages formally. Thus, the language defined is useful to express a specific pattern information. XML abstraction architecture can be summarized in 3 levels (see Figure 2):

- Generic language of XML Schema. It is a meta-data model. It allows describing data descriptions. It defines the meta-language that can be used to construct specific languages.
- Particular language defined in XML Schema. It is a data model. It allows describing data. It defines the language used to describe a particular data.
- XML document. Data description based on the language defined by the schema.

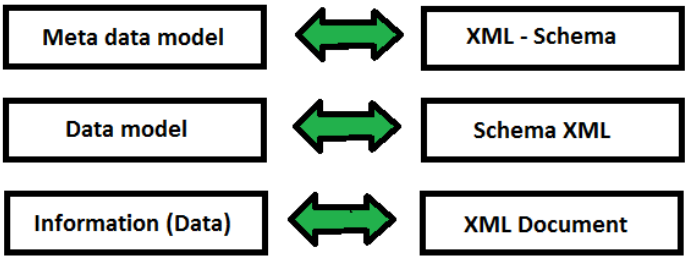


Figure 2: XML transformation

2.5 HTML5

HTML5 (HyperText Markup Language, version 5) is the fifth major revision of the basic language of the World Wide Web [8], HTML. It specifies two variants of syntax to HTML: HTML (text/html), the variant known as HTML5 and XHTML variant known as syntax XHTML5 to be be served as XML (XHTML) (application / xhtml + xml). It was the first time that HTML and XHTML were developed in parallel.

It is still in experimental mode, although it is used by many web developers for its progress, improvements and benefits.

Older versions of browsers do not recognize HTML5 because of its new labels, so they recommend you to upgrade your browser. The development of this code is controlled by the W3C.

HTML5 adds labels to handle the Semantic Web (Web 3.0): header, footer, article, nav, time (date of the content), which can describe what is the meaning of the content.

They have no particular impact on the display, they are aimed at searh engines. The search engines can index and interpret this meta information for not seeking simply occurrences of words in the text of the page. He can incorporate files to pages RDF / OWL (with meta information) to describe relationships between the terms used.

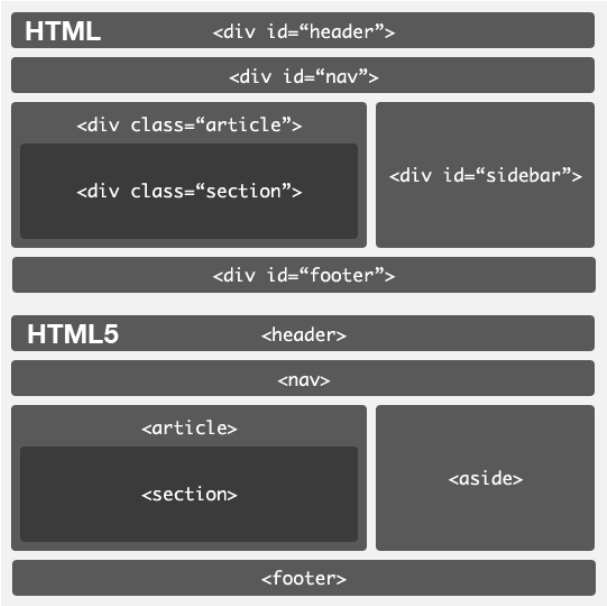


Figure 3: Comparative structure of HTML and HTML5

HTML5 provides a number of new elements and attributes that reflect the typical use of existing web sites [9]. Some of these changes are technically similar to labels textless div textgreater and textless span textgreater, but have a semantic meaning, such as textless nav textgreater (block navigation of the website) and textless footer textgreater. Otros elementos proporcionan nuevas funcionalidades a travΓ©s de una interfaz estandarizada, como los elementos <audio>y <video>. Other elements provide new functionality through a standardized interface, such as items textless audio textgreater and textless video textgreater. See figure 3.

There are improvements in the textless canvas textgreater, capable of rendering 3D elements in some browsers. HTML 4.01 has some obsolete elements, including presentational elements, such as textless font textgreater and textless center textgreater, whose effects are driven by style sheets (CSS). It also highlights the importance of DOM scripting for the behavior of the web.

When browsers render a page, they build a "Document Object Model", a collection of objects that represent HTML elements on the page. Each element (textless p textgreater, textless div textgreater, textless span textgreater, etc..) is represented in the DOM by a different object.

All DOM objects share common characteristics, although some have more than others. In browsers that support HTML5 features, some objects have a single property and watching the DOM you know the features the browser supports.

3. IMPLEMENTATION

The most important pieces to develop desktop applications using Visual C # are the forms. A form is a blank page that the developer fills with controls to create a user interface, and code to process the data. Objects form contains controls. Each type of control has its own set of properties, methods and events that make it suitable for a particular purpose. You can manipulate the controls and write code to add controls dynamically at runtime.

Apart from developing the application in C #, you must develop a XML Schema file and a XSLT template.

XML Schema has been created to validate that the XML generated by the application follows a right scheme to avoid errors. As seen in this Master the advantages of using XML Schema with respect to the use of DTDs (Document Type Definition) is that the syntax used is XML and supports the specification of data types and user-defined types and numerical constraint checking.

The construction of XML in the application is made by System.Xml.Linq and System.Xml library. The construction of XML in the application is made inter alia by System.Xml.Linq and System.Xml library. Language-Integrated Query (LINQ) is a set of features introduced in Visual Studio 2008 that adds query capabilities to the syntax of the C# and Visual Basic. Visual Studio includes LINQ provider assemblies that enable the use of LINQ with .NET Framework collections, SQL Server databases, ADO.NET Datasets and XML documents. The library contains the classes listed above for LINQ to XML. LINQ to XML is an XML programming interface that can create and modify XML documents.

With LINQ to XML is possible:

- Load XML from files or streams.
- Serialize XML to files or streams.
- Create XML trees from scratch using functional construction.
- Querying XML Trees through LINQ consultations.
- Manipulating XML trees in memory.
- Validate XML trees using XSD.
- Using a combination of these features to transform XML tree forms.

FXCop has been used to assess the quality of application code.

4 . CONCLUSIONS

One aim of this project was to study the fundamentals of MDA, which I played and written in the chapter entitled "State of the art." Also, in the same paragraph, it has been researched the creation of XSLT templates to use as language processing and generation of HTML5.

For this, we have integrated these templates in a simple application that generates guidebooks, but you can extend the use of it for other purposes. This aim has been detailed in the implementation chapter.

The development of the research allows a series of contributions that are described below:

- Model-driven development is a fact and you can use this architecture to carry out various projects.

- XSLT transformations and the use of markup language allow the development of software in an agile and quality

5. SUGGESTED GUIDELINES

Surely there are many options and details that can be improved or added to the functionality of the system. We can mention:

- Perform a more generic application that could "understand" / "interpret" the markup language (XML), generate the specific application and then just incorporate the XSLT template.
- Develop an application to obtain a XML document markup that matches XSLT from UML.
- Develop a web application or a mobile application. From a business point of view, develop a mobile application where the user can create their own travel guide. Create the same application for social networks, for example Facebook, and share travel guides.
- Add new structures - attributes to the XSLT template that generates HTML5
- Create an application where the elements are drag and drop.
- Generate documents in other formats besides HTML, such as PDF.

6. REFERENCES

[1] *HONGMING, Liu and LIZHANG, Qin and XIAOPING, Jia*, Model transformation based on meta templates.
International Conference on Software Engineering Research & Practice. 2006.
De Paul University. School of Computer Science, Telecommunications and Information Systems.
[2] *CABOT, Jordi*, RelaciГin entre MDA, MDD y MDE
http://modeling-languages.com/es/blog/content/relaci-n-entre-mdamdd-y-mde. 2009.
[3] OMG, http://www.omg.org
[4] *SCHMIDT, Douglas*, Model-Driven Engineering
Vanderbilt University. 2006.
[5] *VALLECILLO, Antonio*, Model Driven Development
Universidad de MГЎlaga. 2003.
[6] *PEГ‘A, Juan Carlos*, TicXML: Generando diferentes interfaces de usuario finales a partir de una Гєnica especificaciГin declarativa
Universidad de Castilla-La Mancha. 2007.
[7] XSLT Specification, http://www.w3.org/TR/xslt20/
[8] HTML5 Specification, http://dev.w3.org/html5/spec/single-page.html
[9] *KJAER, Mads*, HTML 5 and CSS 3: The Techniques YouвЂ™ll Soon Be Using, http://net.tutsplus.com, 2009

Anexo C
Manual de usuario

El objeto de este documento es recoger el uso y descripción de funcionamiento de la aplicación.

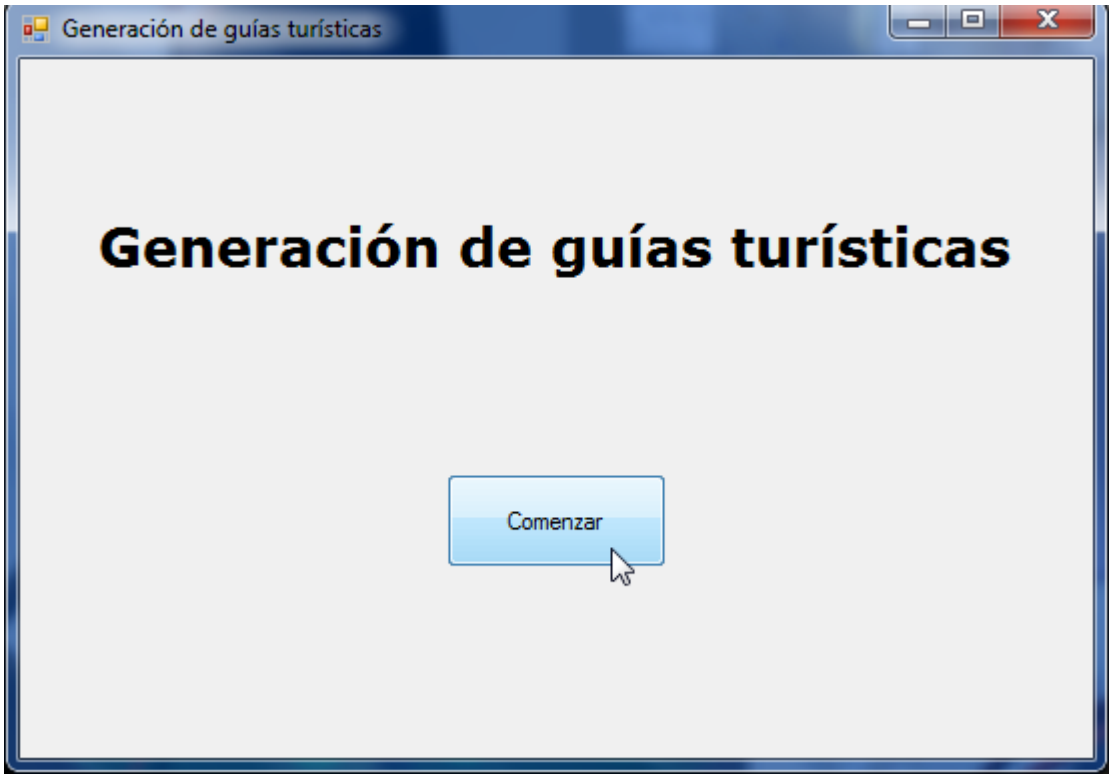
C.1. INSTALACIÓN Y EJECUCIÓN

Debe de descomprimir el archivo que contiene la aplicación en la carpeta que desee. Una vez descomprimido, se debe ejecutar el fichero GuiaTurismo.exe

C.1.1. Pantallas

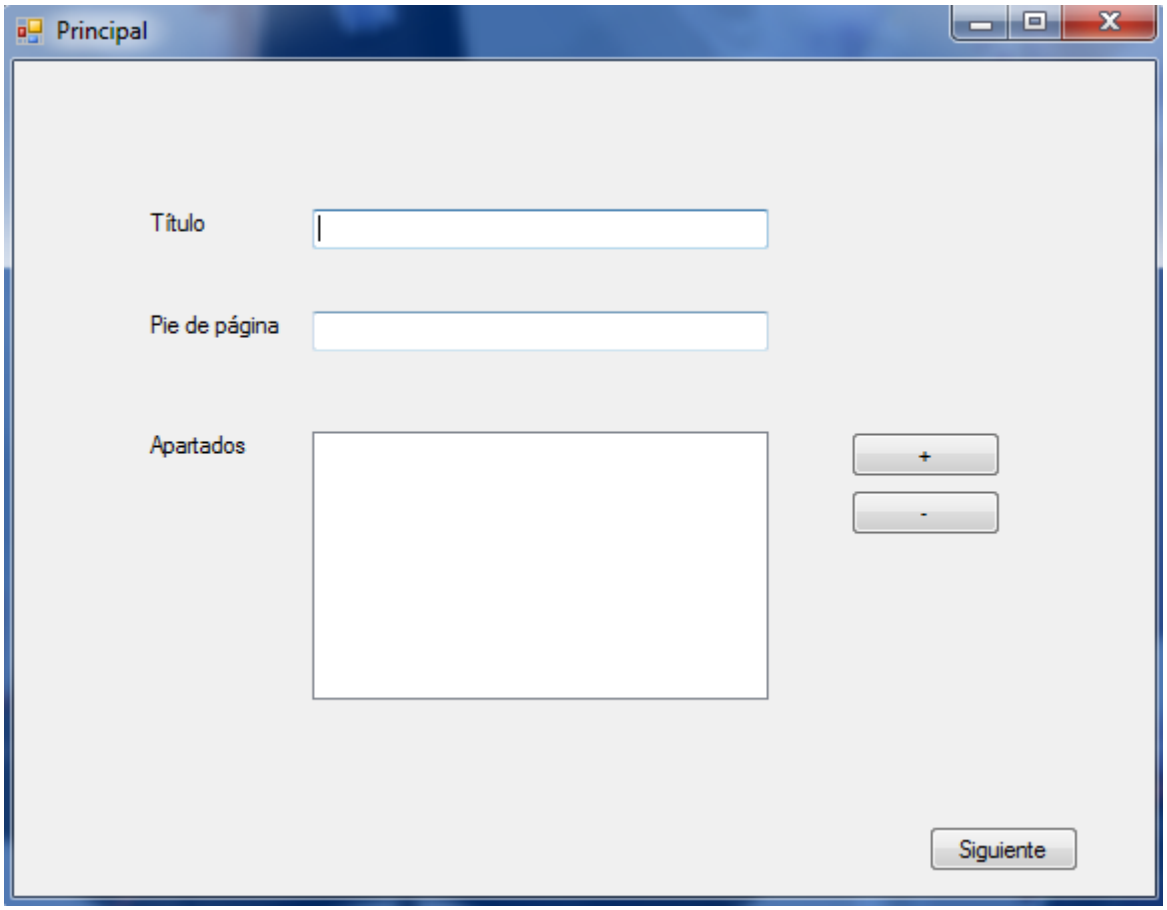
C.1.1.1 Pantalla de acceso

Una vez ejecutado el fichero GuiaTurismo.exe aparecerá la pantalla principal del programa. Se debe pulsar el botón "Comenzar" para iniciar el proceso de creación de la guía turística.

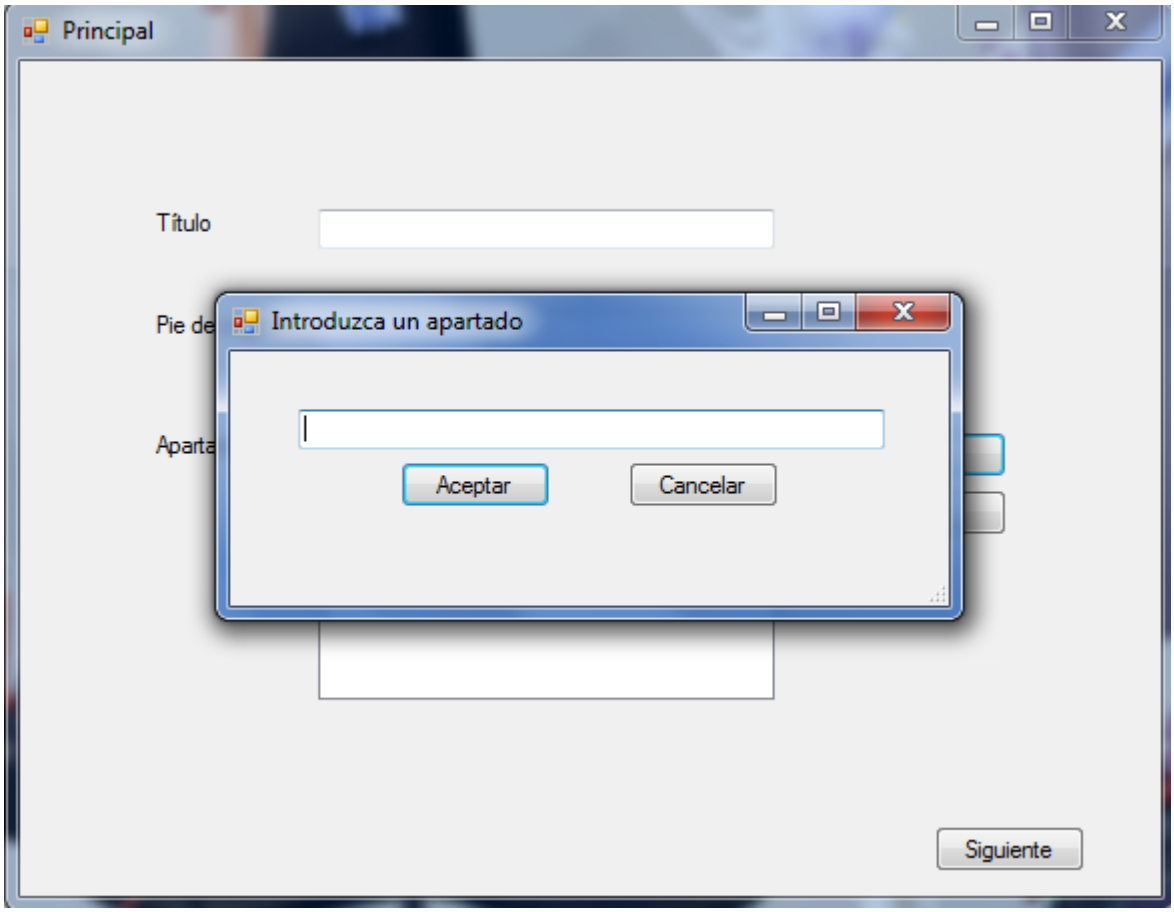


C.1.1.2 Pantalla principal

En esta pantalla se debe de rellenar el título de la guía y los apartados que contendrá. También es posible aunque no obligatorio rellenar el pie de página que aparecerá en la página Web.

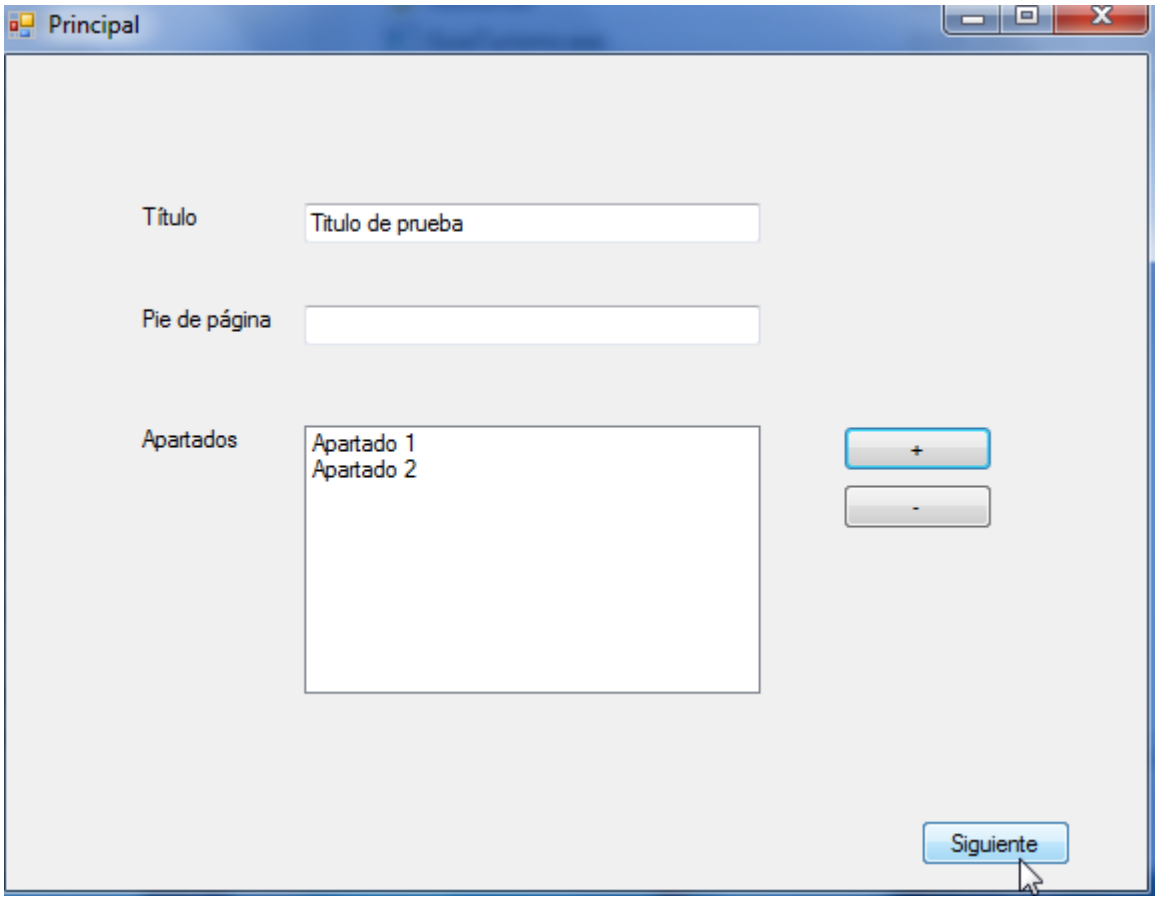


Para introducir un apartado se debe pulsar sobre el botón “+”. Una vez pulsado aparecerá una pantalla emergente para introducir el título del apartado.



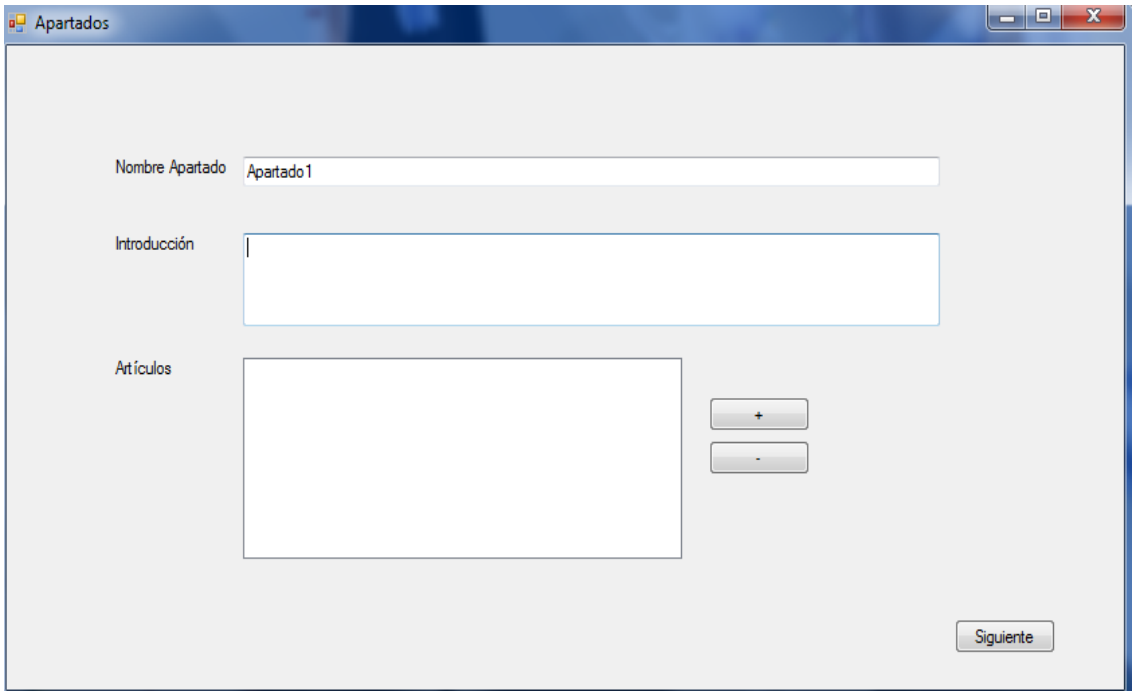
Cabe la posibilidad de eliminar un apartado seleccionando el apartado en la lista y pulsado el botón "-".

Una vez relleno todos los apartados deseados se debe pulsar el botón "Siguiente" para continuar con la creación de la guía.



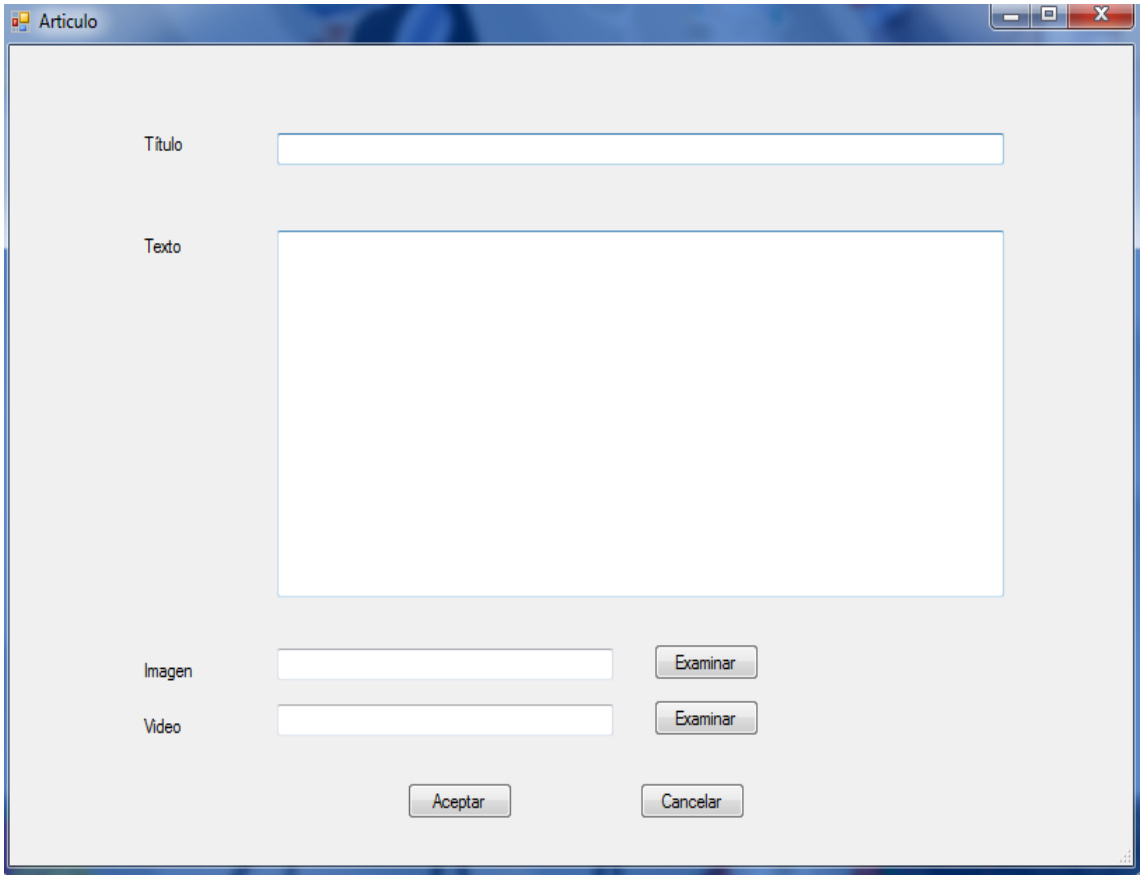
C.1.1.3. Pantalla de apartados

Por cada uno de los apartados que se hayan introducido en la pantalla previa debemos de ir rellenando cada uno de los mismos.



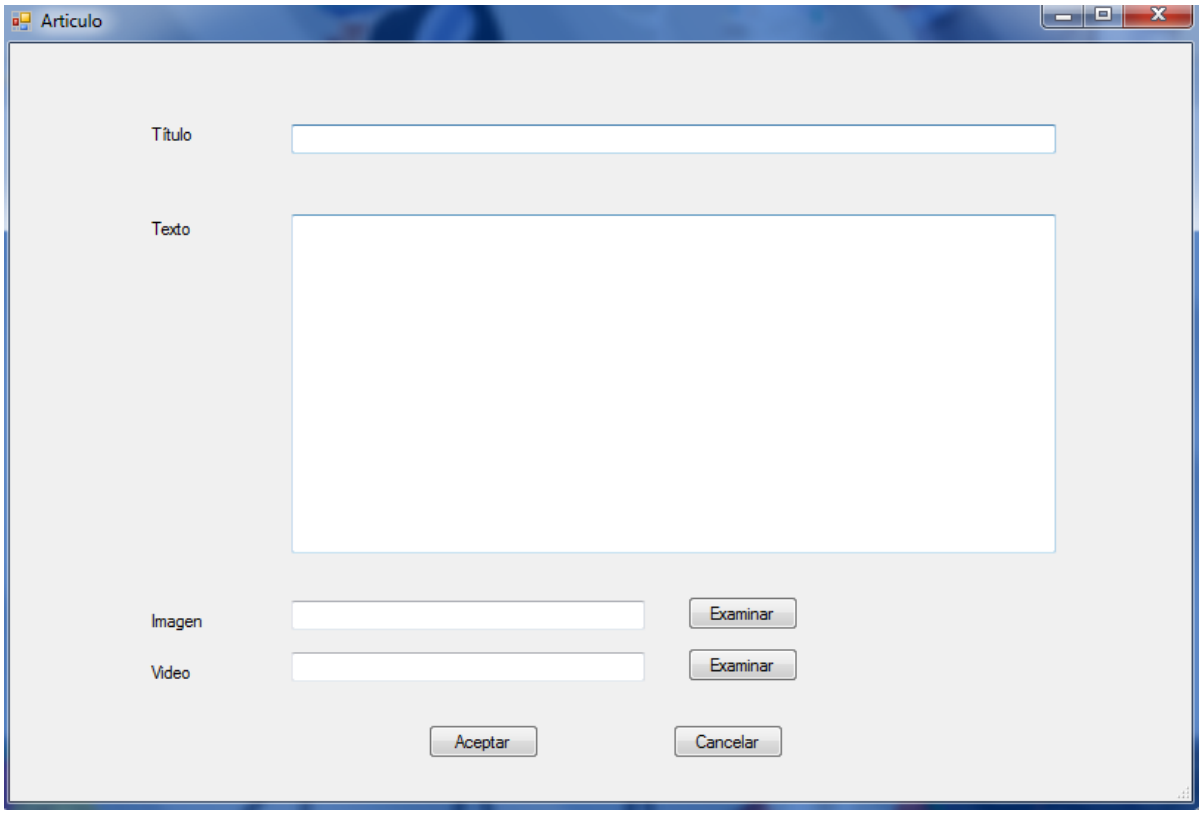
Para ello debemos de rellenar una introducción del apartado y añadir los artículos que se desean (de manera similar a la introducción de apartados en la pantalla previa).

Para el relleno de los artículos es preciso pulsar el botón '+'. También cabe la posibilidad de eliminar artículos pulsando el botón "-".



C.1.1.4. Pantalla de artículos

Esta pantalla nos permite introducir un artículo a un apartado.



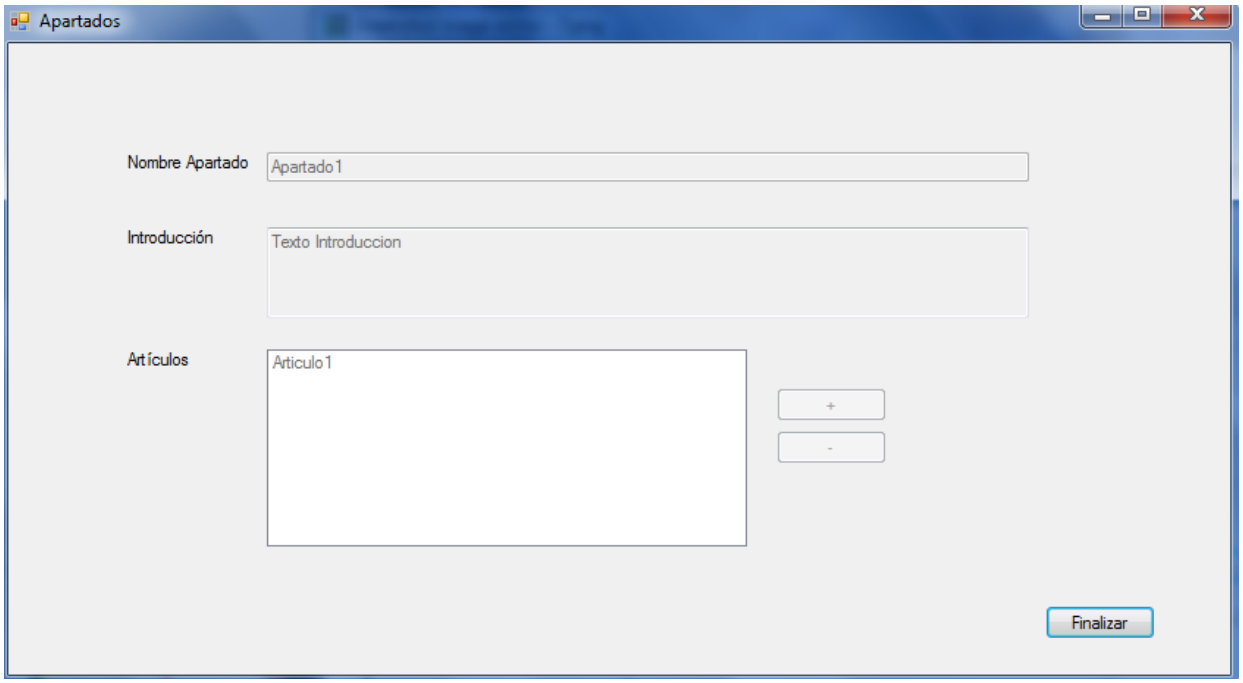
Se debe rellenar el título del artículo y el contenido de texto. También cabe la posibilidad de seleccionar una imagen y/o un vídeo. Para seleccionar una imagen o vídeo es necesario pulsar el botón “Examinar”. Realizada esta acción aparecerá una pantalla emergente que da la posibilidad de elegir un archivo de imagen o vídeo en los formatos soportados.

Los formatos de imagen soportados son .jpg, .jpeg, .png, . gif y .bmp. El formato de vídeo soportado es .mp4. Se recomienda guardar todas las imágenes y vídeos en una carpeta que cuelgue de la carpeta “Resources” de la aplicación.

Una vez rellenado el artículo es posible pulsar el botón “Aceptar” para guardar los cambios o “Cancelar” para salir sin guardar el artículo.

C.1.1.5. Pantalla de finalización

Una vez finalizada la introducción de información de todos los apartados, aparecerá en la parte inferior derecha de la aplicación el botón “Finalizar”.



Una vez pulsado el botón “Finalizar” si todo está correcto se generará un archivo “index.html” en la carpeta “Resources” de la aplicación con la guía generada.