Universidad de Valladolid

E. T. S. DE INGENIERÍA INFORMÁTICA

Ingeniero en Informática

Virtual Museum sobre plataforma Java EE

Alumno: ***Francisco José Pascual Martínez***

Tutores: ***Jesús Vegas Hernández***
Pablo de la Fuente Redondo

Índice de Contenidos

ÍNDICE DE CONTENIDOS

ÍNDICE DE CONTENIDOS

ÍNDICE DE CONTENIDOS

Índice de figuras

Lista de Tablas

PARTE I

INTRODUCCIÓN Y ESTUDIO PREVIO

PREFACIO

El trabajo que comienza aquí tiene su eje central en el concepto de Museo Virtual. Al parecer, los museos virtuales existen gracias a dos ideas provocadoras lanzadas en el siglo pasado por dos franceses. Por una parte, la idea de Marcel Duchamp respecto a la creación de un museo transportable (*boîte-en-valise* 1 - un maletín con reproducciones de sus obras en miniatura). Y por otra, la idea de un museo imaginario de André Malraux. Museo transportable (movilidad, acceso remoto) y museo imaginario (inmaterialidad, virtualidad física) presentan hoy una clara relación con el actual término de Museo Virtual. Tradicionalmente el concepto de museo ha denotado un espacio arquitectónico con unas fronteras físicas bien definidas en cuyo interior se accede a sus colecciones, pero en la actualidad un museo expande su presencia y trasciende las barreras físicas gracias al desarrollo de las tecnologías de la información, crecimiento de Internet y el deseo de concebir nuevos espacios para la difusión, creación de arte, y comunicación bidireccional con el público en general y otros sistemas. Gracias a esta expansión, podemos encontrar museos virtuales no sólo de entidades emblemáticas o que gozan de gran reconocimiento, sino de personas desconocidas que montan su galería con reproducciones de sus obras, hasta museos en el ciberespacio que exhiben obras digitales que no tienen correspondencia con una obra física.

Figura 1: Boîte-en-valise de Marcel Duchamp

Capítulo 1

MARCO GENERAL DEL PROYECTO

Gracias al telégrafo, todos los habitantes de la Tierra podrán convivir en un solo vecindario intelectual.

General Alonzo Jackman

Índice del Capítulo

1.1. Justificación inicial

Uno de los fenómenos más interesantes de los últimos años en el mundo del arte ha sido la extensión de la red de museos de arte en el mundo y sobre todo en el Norte de Europa, España y América del Norte, también en Asia. El creciente deseo de acceder a la cultura (y en concreto al conocimiento del arte) por parte del público medio de la sociedad actual es una realidad palpable que ha provocado la creación y apertura de decenas de nuevos museos en los últimos años.

Las nuevas tecnologías de la información ofrecen a los museos una oportunidad hasta ahora desconocida para responder a los requerimientos de la sociedad. Desde el momento en que fue posible almacenar, procesar y recuperar texto, sonido e imágenes fijas y en movimiento, situarlas en redes y enviarlas a cualquier punto del globo a cualquier hora del día, el acceso a los museos empezó a tomar una dimensión diferente. Además de la utilización tradicional, el arte en Internet ofrece nuevas posibilidades como la interactividad y la desaparición de las barreras físicas.

Dado el auge de los museos virtuales, parece interesante el planteamiento de un sistema informático que asista en la creación y difusión de museos virtuales, basándose en diversos criterios como el uso, contenido, diseño de interfaz, interoperabilidad con otros sistemas, etc.

1.2. Objetivos del proyecto

El presente proyecto tiene su motivación original en la evolución de otro proyecto fin de carrera previo ([Pér03]). En él, el objetivo principal era desarrollar un sistema de configuración de museos virtuales de arte, que pudiesen ser visualizados tanto en un navegador web como a través de un dispositivo móvil (PDA, teléfono, etc). Siguiendo esta línea original, el proyecto que nos ocupa va a establecer sus objetivos como una evolución de las premisas iniciales del proyecto anterior e incorporación de otras nuevas relacionadas principalmente con los nuevos enfoques, técnicas y conceptos de la Web.

A partir de las funcionalidades que ofrecía el proyecto anterior, el presente pretende añadir ciertas características que se perfilan a continuación:

- Dar soporte para agregar metadatos a los recursos que presenta cada museo virtual. Como se verá, el estándar de metadatos a seguir es Dublin Core.
- El sistema resultante debe producir tres subsistemas diferenciados para cada uno de los tres roles generales del sistema: administrador del sistema global, responsable del museo y visitante de los museos virtuales generados por el responsable .
- Brindar un sistema de publicación de contenidos en general flexible para que el responsable pueda componer un sitio completo, en la línea de los habituales gestores de contenidos o CMS (*Content Management Systems*).
- Permitir la personalización de la apariencia de los sitios web generados para cada museo virtual, mediante la elección de temas o plantillas que produzcan una composición diferente en cuanto a contenidos y forma a gusto de cada usuario.
- Producir un sistema muy escalable en cuanto a tecnologías de desarrollo, de forma que a partir de la arquitectura seleccionada cada capa sea independiente del resto, de cara a futuras modificaciones y/o ampliaciones. Esta escalabilidad debería permitir un amplio abanico de cambios entre los que podemos contar: base de datos, contenedor web de la aplicación, gestor de persistencia, lógica de negocio, internacionalización de mensajes, composición y forma visual de las aplicaciones, formato de los museos virtuales generados, etc.
- Orientar los sistemas resultantes a las ideas propuestas por la Web 2.0. En este sentido, se ofrecerán distinas vías de acceso a la información de los recursos para el público en general u otros sistemas entre los que podemos encontrar, además del propio sitio web: Google Maps, Flickr, Picasa, RSS, vista para móviles, RDF o acceso por servicios web.

1.3. Contenido de la memoria

La memoria del proyecto fin de carrera que viene a continuación se ha dividido en las siguientes partes:

- Introducción y estudio previo: se hace una introducción a los museos virtuales a través de su situación actual en Internet (capítulo 2), y a continuación (capítulo 3) se describe el estándar de metadatos seguido en Virtual Museum, partiendo de la Web y Web Semántica, hasta llegar al estándar Dublin Core, pasando por conceptos importantes como museo virtual, biblioteca digital, metadatos, etc.
- Desarrollo del sistema: una vez visto dónde se enmarca el proyecto y los fundamentos subyacentes, se pasa a describir el proceso software seguido a través de sus flujos de trabajo fundamentales. De ahí que en una serie de capítulos sucesivos (a partir del capítulo 4) se reúnan los artefactos resultantes de las etapas de requisitos, análisis, diseño, implementación y pruebas.
- Conclusiones y trabajo futuro: en los dos últimos capítulos, con el sistema construido se comentan diversas reflexiones del producto desarrollado y se evaluan ciertos aspectos del mismo. Además, se ofrece al lector posibles líneas de trabajo futuro que podrían ser interesantes como continuación y ampliación del presente proyecto.
- Apéndices: la última parte de la memoria reúne la información de referencia que pueden servir de consulta adicional respecto de los capítulos anteriores. Hay que hacer notar que también se incluye documentación de apoyo en el CD anexo que por sus características no se han incorporado a la memoria impresa.

Una vez demarcado el ámbito general del proyecto a partir del siguiente capítulo se irán describiendo las ideas principales que han llevado al proyecto desde su planteamiento hasta la puesta en marcha del sistema.

Capítulo 2

PANORÁMICA DE MUSEOS VIRTUALES EN INTERNET

Sin arte la vida sería un error.

Friedrich Wilhelm Nietzsche

Índice del Capítulo

2.1. Situación de los museos en España

Antes de afrontar el análisis del sistema a realizar, es necesario enmarcar el proyecto en la situación actual de los museos virtuales, o museos en general. Para conocer mejor la realidad relativa

a museos y colecciones museográficas se puede acudir a las estadísticas ofrecidas por el Ministerio de Cultura. Desde su sitio web (*http://www.mcu.es/estadisticas/MC/EM/index.html)* se puede obtener información acerca de la oferta museística clasificando los resultados por ámbito territorial y tipología del museo. Aunque el volumen de tablas y gráficos es amplio, de cara al presente proyecto, se tendrán en cuenta aquellos indicadores más específicos como la disponibilidad, características y objetivos de la página web. Por detallar un poco más estos indicadores se pueden mencionar:

- Página web propia
- Comunicación con el visitante (en ambos sentidos)
- Servicios disponibles online
- Alojamiento en servidor propio
- Conexiones con portales de otros museos
- Visita virtual

El cuadro 2.1 muestra la situación en España del año 2006 para los indicadores mencionados. En él se puede apreciar que la presencia web de los museos censados en España sobrepasa ligeramente el 50 % y que en cuanto a cada uno de los indicadores mostrados a continuación se obtienen porcentajes relativamente bajos. El cuadro 2.2 desglosa los resultados anteriores por comunidad autónoma.

Indicador	Porcentaje (sobre 1343 museos)
Página web	57.2 %
Servidor Propio	13.3 %
Conexión a portales	21.5 %
Visita Virtual	14.4 %
Comunicación del museo al visitante	21 %
Comunicación del visitante al museo	13.3 %
Servicios online	7.1 %

Cuadro 2.1: Estadísticas Museos y Colecciones Museográficas

Por otra parte aunque este proyecto se plantea como una herramienta para la generación de museos virtuales de cualquier naturaleza, puede ser interesante mostrar (en el cuadro 2.3) los valores relativos de museos según su tipología.

A la vista de los resultados anteriores, se puede pensar que la apuesta por la presencia web de los museos en España es decidida pero en algunos casos esta incorporación a Internet en cuanto a una experiencia más rica es relativamente tímida. Por eso, el proyecto que abordamos representará un paso más en este proceso para acercar la realidad museística al visitante en general mediante el uso de las nuevas tecnologías por la facilidad de acceso que supone y porque amplía sin límites el público potencial más allá de sus puertas físicas.

2.2. Museos Virtuales en la Red

Actualmente, disponemos de una cantidad cada vez mayor de museos que ofrecen al público variada información a través de Internet. Si bien no hemos hecho una revisión pormenorizada, a con-

Comunidad Autónoma	Página web (%)	Servidor Propio (%)	Conexión a portales (%)	Visita Virtual(%)
Andalucía	70.3	18.6	24.9	26.9
Aragón	61.3	14.5	11.3	11.3
Asturias (Principado de)	63.9	13.9	36.1	25.0
Baleares	48.5	15.2	9.1	6.1
Canarias	52.7	7.3	16.3	10.9
Cantabria	66.7	-	41.7	33.3
Castilla y León	43.6	13.4	7.6	9.9
Castilla La Mancha	29.0	9.9	4.6	7.6
Cataluña	97.1	-	90.2	-
Comunidad Valenciana	39.9	10.4	9.2	12.9
Extremadura	55.3	8.5	14.9	12.8
Galicia	53.7	13.4	38.8	10.4
Madrid (Comunidad de)	76.5	30.4	18.6	29.4
Murcia (Región de)	62.0	11.3	21.1	15.5
Navarra (Comunidad Foral de)	53.8	23.1	19.2	3.8
País Vasco	78.3	13.0	15.9	29.0
Rioja (La)	37.5	12.5	25.0	-
Ceuta	66.7	33.3	-	-
Melilla	33.3	33.3	-	16.7

Comunidad Autónoma	Comunicación al visitante(%)	Comunicación al museo(%)	Servicios online(%)
Andalucía	29.7	17.2	14.5
Aragón	24.2	17.7	12.9
Asturias (Principado de)	36.1	25.0	8.3
Baleares	1.5	-	-
Canarias	25.5	14.5	10.9
Cantabria	25.0	16.7	8.3
Castilla y León	17.4	12.2	5.2
Castilla La Mancha	9.9	7.6	3.1
Cataluña	-	-	-
Comunidad Valenciana	22.1	16.0	7.4
Extremadura	17.0	8.5	-
Galicia	11.9	4.5	1.5
Madrid (Comunidad de)	34.3	14.7	9.8
Murcia (Región de)	19.7	16.9	2.8
Navarra (Comunidad Foral de)	26.9	19.2	3.8
País Vasco	56.5	37.7	23.2
Rioja (La)	25.0	12.5	12.5
Ceuta	-	-	-
Melilla	16.7	16.7	16.7

Cuadro 2.2: Estadísticas de Museos y Colecciones Museográficas por Comunidad Autónoma

tinuación resumimos aquellos que por alguna característica especial (contenido, servicio, localización física, idiomas, interoperabilidad, etc) caben destacarse para el desarrollo del proyecto. También incluimos la organización de contenidos que se presenta a primera vista a través de su menú principal para intentar plantear una jerarquía de contenidos común en nuestro proyecto.

2.2.1. Louvre

www.louvre.fr (París)

- Organización de contenidos: museo, obras, exposiciones, auditorio, actividades, recorridos, enlaces directos.
- Selección de idioma (francés, ingles y japonés)

Indicador	Porcentaje (sobre 1343 museos)
Arqueológico	%
Arte contemporáneo	%
Artes decorativas	%
Bellas artes	%
Casa-museo	%
Ciencia y tecnología	%
Ciencias naturales e historia natural	%
De sitio	%
Especializado	%
Etnografía y antropología	%
General	%
Historia	%
Otros	%

Cuadro 2.3: Estadísticas de Museos y Colecciones Museográficas por Tipología

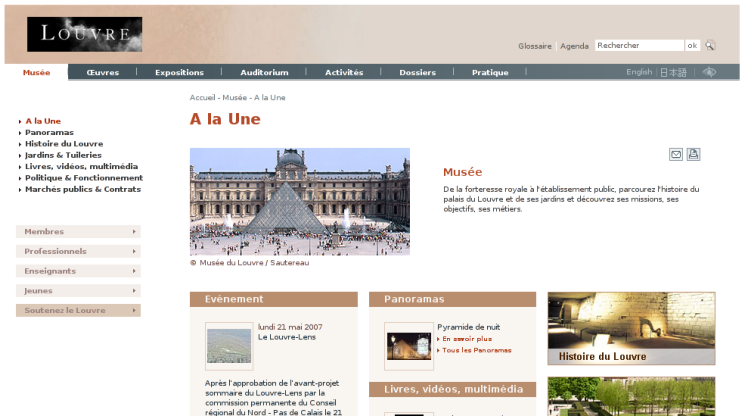


Figura 2.1: Louvre

- Es frecuente poder acceder a todos los contenidos de una categoría desde el menú, aunque también se puede acceder a los contenidos particulares de cada categoría desde el propio menú.
- Ofrece información bastante detallada de una serie obras elegidas dentro de cada temática.

2.2.2. Instituto Valenciano de Arte Moderno

www.ivam.es (Valencia)

- Organización de contenidos: presentación, fondos, exposiciones, biblioteca, publicaciones, actividades, servicios, didáctica, recorrido virtual y enlaces externos.
- Selección de idioma (castellano, valenciano e inglés)
- Presenta una página de inicio distinta al resto para presentar al museo y resaltar ciertos contenidos.

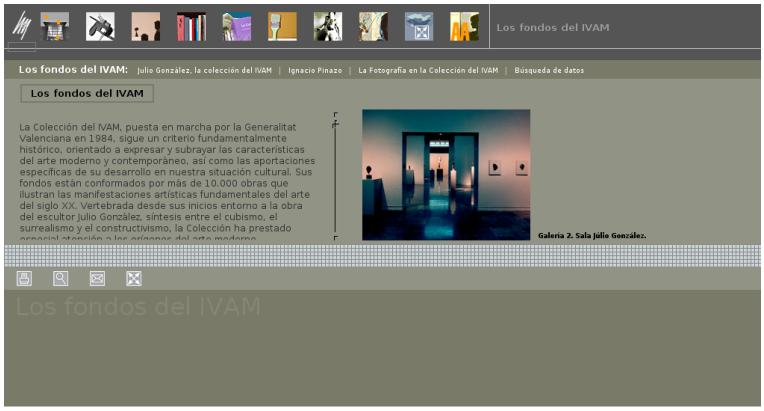


Figura 2.2: Instituto Valenciano de Arte Moderno

2.2.3. Museo Patio Herreriano

www.museopatioherreriano.org (Valladolid)

Figura 2.3: Museo Patio Herreriano

- Organización de contenidos: información del museo, información general, colección, exposiciones, educación, centro de documentación, audivisual, arte en la red y publicaciones.
- Presenta información tanto acerca de la obras que mantiene como de sus autores.
- La página de inicio lista las novedades del museo en una apariencia bastante similar a la que ofrece un blog (fecha, título, texto introductorio y enlace que amplía la información).

2.2.4. Fundación Colección Thyssen-Bornemisza

www.museothyssen.org (Madrid)

Figura 2.4: Fundación Colección Thyssen-Bornemisza

- Organización de contenidos: colección, exposiciones, actividades, información, quiénes somos, colaboradores, tienda.
- Disponibilidad de canales RSS para ciertos contenidos.
- Selección de idioma (castellano e inglés)
- Utilización de recursos multimedia en ocasiones puntuales aunque de diversa índole: audios mp3, presentaciones flash, applets java, etc.
- Existencia de una sección de obras maestras que permite al usuario visualizar las obras más importantes del museo de una forma rápida y sencilla.

2.2.5. Museo de arte metropolitano

www.metmuseum.org (Nueva York)

- Organización de contenidos: obras, tienda, miembros, donaciones, planear visita, calendario, claustros, auditorios y ponencias, recursos educativos, eventos, personalización por usuario (*My Met Museum*), sala de presa y podcast.
- Organización destinada a un amplio conjunto de visitantes con diversos fines.
- Información textual y gráfica muy rica y extensa.

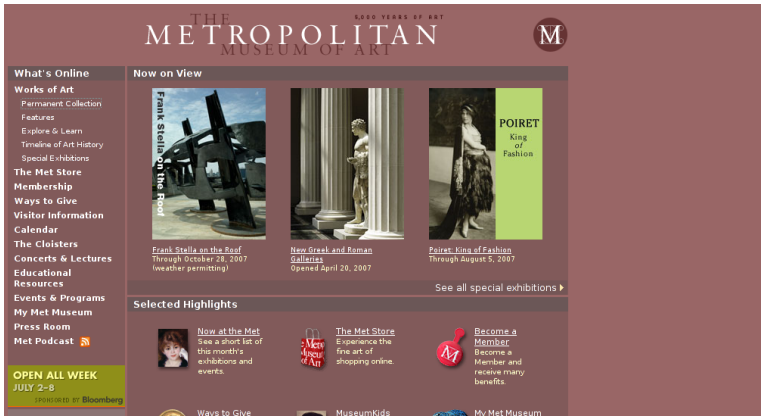


Figura 2.5: Museo de arte metropolitano de Nueva York

- Aunque la apariencia de todo el sitio es uniforme, ciertas zonas o contenidos presentan un estilo particular.

2.2.6. Casa de las Ciencias

casadelasciencias.logro-o.org (Logroño)

Figura 2.6: Casa de las Ciencias de Logroño

- Organización de contenidos: historia, programación, galería de fotos, espacios, normativa, foros de debate, suscripción, información general, centros docentes y publicaciones.
- Presentación clara y sencilla de rápido acceso a sus contenidos.
- Poca profundidad del árbol del mapa web del sitio.

2.2.7. Fundación del Patrimonio Histórico de Castilla y León

www.fundacionpatrimoniocyl.es (Castilla y León)

Figura 2.7: Fundación del Patrimonio Histórico de Castilla y León

- Organización de contenidos: presentación, información general, restauración, arqueología e historia, rutas de turismo cultural, formación y cursos, publicaciones, difusión cultural, prensa, galería de imágenes, enlaces y concursos.
- Estilo uniforme a lo largo de todo el sitio, aunque el contenido de la portada se enfoca en presentar sus últimas novedades.
- Selección de idioma (castellano, inglés y francés).

2.2.8. Museo Nacional de Escultura

museoescultura.mcu.es (Valladolid)

- Organización de contenidos: información general, colección, edificios, atención al público, actividades, publicaciones, servicios, enlaces y mapa.
- Selección de idioma (castellano, inglés y francés).
- Estilo uniforme a lo largo de todo el sitio.

2.2.9. Ciudad de las Artes y las Ciencias

www.cac.es (Valencia)

Figura 2.8: Museo Nacional de Escultura

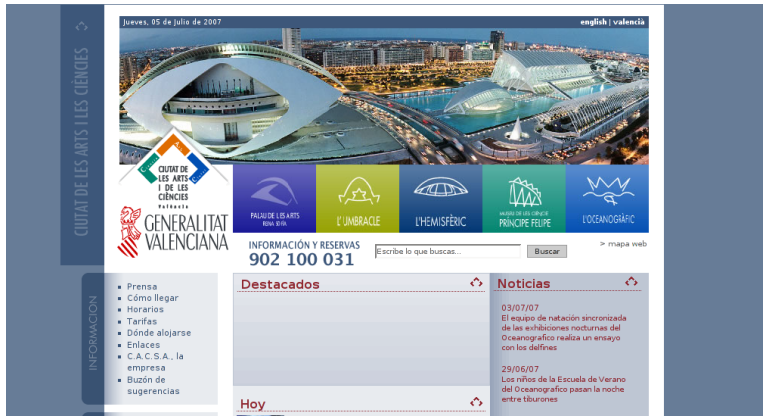


Figura 2.9: Ciudad de las Artes y las Ciencias de Valencia

- Organización de contenidos: prensa, cómo llegar, horarios, tarifas, enlaces, programación, actividades, servicios, visita, galería de imágenes y microsites.
- Selección de idioma (castellano, inglés y valenciano).
- Abundante información distribuida por los distintos públicos, espacios propios y actividades.

2.2.10. Museo de la Ciencia

www.museocienciavalladolid.es (Valladolid)

- Organización de contenidos: museo, información y servicios, exposición permanente, la casa del río, exposiciones temporales, educación, planetario, otras actividades y noticias.

Figura 2.10: Museo de la Ciencia de Valladolid

- Galerías de imágenes directas que presentan el conjuntos de miniaturas (*thumbnails*) pudiendo ampliarla en una ventana *popup*.
- Las colecciones o exposiciones se presentan frecuentemente con un enlace dentro del sitio que redirecciona a otro fuera del museo.

2.2.11. Museo de la Universidad de Valladolid

www3.uva.es/muva (Valladolid)

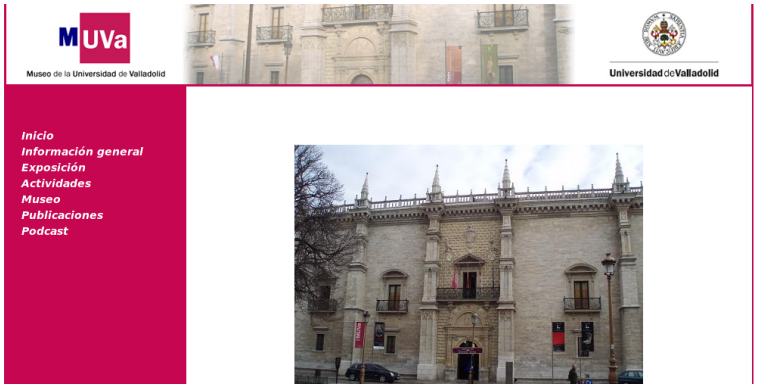


Figura 2.11: MUva

- Organización de contenidos: Información general, exposición, actividades, museo, publicaciones, podcast.

- Presentación diáfana y directa de contenidos.
- Cuenta con la posibilidad de descarga de recursos multimedia bien directamente o bien por suscripción RSS.

2.2.12. Art.Blogging.LA

art.blogging.la (Los Angeles)

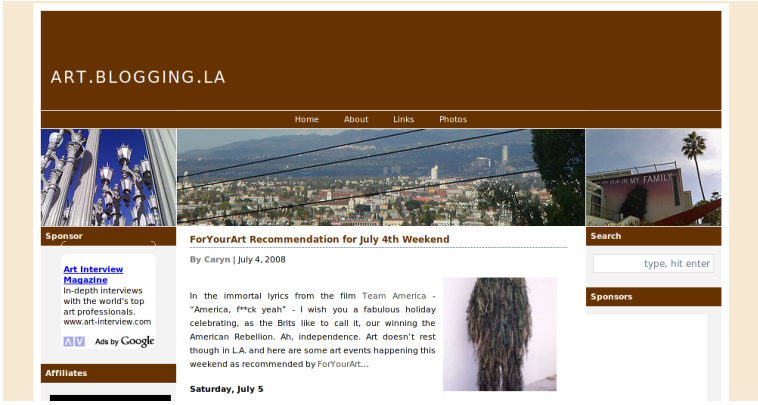


Figura 2.12: Art.Blogging.LA

- Organización de contenidos: Enlaces, fotos, editorial, eventos, exhibiciones, imágenes, entrevistas, noticias, avances (preestrenos), críticas.
- Está basado en *WordPress* por lo que su apariencia, colocación de contenidos e interfaz es la propia de un blog: se basa en contenidos como si fueran *posts* y se pueden presentar agrupados por categorías o por fecha de publicación.

2.2.13. Cocinalia.eu

cocinalia.eu (—)

- Organización de contenidos: categorías, comer en, enlaces, archivos, meta
- Como el caso anterior, está basado en *WordPress* por lo que comparte la orientación de contenidos a *posts*.
- No está asociado a una empresa pública ni privada (al menos no aparentemente).
- Permite añadir comentarios a los contenidos con lo que favorece la comunicación con el visitante, haciendo que éste sea parte activa del mismo.

Figura 2.13: Cocinalia

2.3. Requisitos de interacción

Teniendo en cuenta estos museos, en el presente proyecto se van a tener en cuenta los siguienes aspectos de diseño y de experiencia visual para los museos virtuales que va a ser capaz de generar el sistema:

- Organización de contenidos: Información del museo, información general, localización física (ubicación), obras, autores, recorridos, novedades y eventos. Como hemos visto, esta clasificación puede resultar insuficiente para algunos casos, por lo que sería deseable contar un contenido genérico que no pertenezca a ninguna de las categorías planteadas. Podría tenerse en cuenta una configuración libre de categorías por parte del responsable del museo pero uno de los objetivos primordiales será buscar un manejo fácil e intuitivo de la aplicación diseñada para él.
- Esta organización de contenidos presentada aunque podría llevar a que fuese un reflejo del menú principal del sitio, se observa que el menú principal de cada sitio es lo suficientemente particular y centrado en los propios intereses del museo, que no se puede plantear un menú único y general para los museos virtuales generados.
- Inclusión de múltiples tipos de contenidos multimedia.
- Personalización de la apariencia del sitio publicado; cada museo presenta un diseño diferente ya que forma parte de su propia identidad.
- La selección de idioma no es un servicio disponible generalizado en los museos. Y teniendo en cuenta sólo aquéllos que sí lo ofrecen, el conjunto de idiomas varía de un museo a otro.

Capítulo 3

WEB SEMÁNTICA Y MUSEOS VIRTUALES: DUBLIN CORE

Invertir en conocimientos produce siempre los mejores beneficios.

Benjamin Franklin

Índice del Capítulo

3.1. Introducción

Una cuestión importante a la hora de desarrollar un sistema gestor de museos virtuales es que tiene que englobar dos conjuntos de servicios principales. Por una parte como *biblioteca digital* debe permitir conservar, catalogar, acceder y gestionar el material digitalizado; y por otra parte como *museo virtual* propiamente dicho debe ofrecer a los usuarios exposiciones de dicho material, y otros contenidos interesantes para el visitante por lo que de cara al responsable del museo virtual se debe ofrecer un sistema de gestión de contenidos en la línea de los habituales gestores de contenidos (CMS - *Content Management Systems*) para la creación de sitios web genéricos.

En este capítulo nos centraremos en la idea de biblioteca digital partiendo de conceptos generales como web semántica, metadatos, ontologías, agregación de contenidos, interoperabilidad, etc., llegando a una iniciativa en particular ampliamente usada denominada *Dublin Core*.

3.2. World Wide Web

Aunque la existencia de la Web puede parecer datar de los principios de la historia de los computadores[1] comenzó como un concepto de Tim Berners-Lee ayudado por Robert Cailliau, mientras trabajaban para el CERN (*Centre Européene pour la Recherche Nucléaire*), en Ginebra (Suiza). A Berners-Lee se le ocurrió que podía adaptar un programa que había creado para su uso particular, el ENQUIRE, a las necesidades del CERN de disponer de un sistema para acceder a la enorme y diversa cantidad de información que había en sus sistemas informáticos. Se trataba de un sistema de hipertexto para compartir información basado en Internet. Dicho sistema permitía incoporar multimedia e hipertextos en Internet, almacenando piezas de información y enlazándolas entre ellas. Enquire se ejecutaba en un entorno multiusuario y permitía acceder a varias personas a los mismos datos. En 1989 Berners-Lee entregó su propuesta a varios científicos del CERN pero no obtuvo respuesta. Fue Robert Cailliau quien acudió en su ayuda. Reescribió la propuesta de Berners-Lee en términos que a él le pareció que tendrían más efecto y buscó ayudantes estudiantes y becarios, dinero, máquinas

[1] La idea subyacente de la Web se remonta a la propuesta de Vannevar Bush en los años 40 sobre un sistema similar y posteriormente a Ted Nelson en los años 50 que realiza la primera referencia a un sistema de hipertexto, pero no es hasta 1980, donde cobrará un soporte operativo tecnológico real para la distribución de información en redes.

y espacio en oficinas para poder trabajar. Así, en septiembre de 1990 recibieron el visto bueno y los dos comenzaron a escribir el nuevo sistema de hipertexto, dando origen a la Web como hoy la conocemos.

El concepto, subyacente y crucial, del hipertexto tiene sus orígenes en viejos proyectos de la década de los 60, como el Proyecto Xanadu de Ted Nelson y el sistema on-line NLS de Douglas Engelbart. Los dos, Nelson y Engelbart, estaban a su vez inspirados por el proyecto nunca materializado MEMEX, de Vannevar Bush. El gran avance de Berners-Lee fue unir hipertexto e Internet. En su libro *Weaving the Web*, explica que él había sugerido repetidamente que la unión entre las dos tecnologías era posible para miembros de las dos comunidades tecnológicas, pero como nadie aceptó su invitación, decidió, finalmente, hacer frente al proyecto él mismo. En el proceso, desarrolló un sistema de identificadores únicos globales para los recursos web: el Uniform Resource Identifier (URI).

World Wide Web tenía algunas diferencias de los otros sistemas de hipertexto que estaban disponibles en aquel momento:

- WWW sólo requería enlaces unidireccionales en vez de los bidireccionales. Esto hacía posible que una persona enlazara a otro recurso sin necesidad de ninguna acción del propietario de ese recurso. Con ello se reducía significativamente la dificultad de implementar servidores web y navegadores (en comparación con los sistemas anteriores), pero en cambio presentaba el problema crónico de los enlaces rotos.
- A diferencia de sus predecesores, como HyperCard, World Wide Web era no-propietario, haciendo posible desarrollar servidores y clientes independientemente y añadir extensiones sin restricciones de licencia.

Su diseño tenía una base técnica relativamente sencilla, lo que ayudó a que esta tecnología ganará en popularidad por las masas. Berners-Lee quería que cualquiera pudiera colocar información en un ordenador y hacer esta información accesible a cualquier otra persona, en cualquier parte. Esperaba que eventualmente, las máquinas pudieran ser capaces de usar esta información en la Web. Y en último término, pensaba que esto permitiría una colaboración potente y efectiva humano-máquina-humano:

> ”Siempre he imaginado el espacio de la información como algo a lo que cualquiera tiene un acceso inmediato e intuitivo, y no sólo navengando sino creando...
> Las maquinas llegarán a ser capaces de analizar toda la información en la Web (el contenido, enlaces y las transacciones entre las personas y máquinas.
> ... cuando [la Web Semántica] emerja, los mecanismos del día a día del trato, burocracia y nuestras vidas diarias serán asistidas por máquinas que hablen con otras máquinas, dejando a las personas que provean la inspiración y la intuición”
> (Tim Berners-Lee, 2000)

De forma gradual, otros hackers se sumaron a su esfuerzo proporcionando realimentación, estímulo, aportaciones de código fuente y apoyo moral. A medida que el grupo fue ampliándose, Berners-Lee organizó una comunidad similar a la Internet Society de Vinton Cerf, el World Wide Web Consortium, en un esfuerzo por impedir y prevenir la absorción comercial de la red mundial de

redes. Berners-Lee rechazó de plano todas las ofertas comerciales recibidas, rasgo que uno de sus amigos considera significativo de su mentalidad:

> A medida que tecnólogos y empresarios lanzaban o fusionaban compañías para explotar Internet, parecían quedarse anclados en la pregunta de "¿Cómo puedo hacer mía la Red?". En cambio, Tim se preguntaba: "¿Cómo puedo hacer vuestra la Red?"

Técnicamente, se podría decir que la web propiamente dicha está basada en cuatro pilares o estándares:

- HTML (*Hypertext Markup Language*)
- HTTP (*Hypertext Transfer Protocol*)
- Tipos de datos MIME
- Las localizaciones URL (*Uniform Resource Locator*)

HTML es un lenguaje de marcas que sirve para describir la estructura y la apariencia de los documentos de texto en Internet. Por su parte, una URL proporciona un mecanismo simple de direccionamiento que permite a la web enlazar información de computadores mundialmente. Un ejemplo de URL podría ser *http://www.uva.es/index.html*, donde *www.uva.es* es el nombre de dominio del servidor, *index.html* se "refiere" el nombre del fichero en ese computador, y *http* es el nombre del protocolo. Este protocolo HTTP consiste en una serie de reglas a seguir para el intercambio de mensajes entre dos computadores. Por último, en la web el tipo de datos de un fichero se especifica por un esquema llamado MIME, el cual define el formato para interpretar un archivo. El conjunto de los distintos tipos MIME es un estándar y es usado ampliamente por numerosas aplicaciones, aunque dicho conjunto puede ampliarse usando etiquetas propias.

Para permitir al usuario recuperar y visualizar documentos de hipertexto HTML se empleaban navegadores. La historia de los navegadores ([ADe08]) cuenta con docenas de innovadoras aplicaciones desarrolladas a lo largo de los últimos años por muchas personas y equipos. ViolaWWW fue un navegador bastante popular en los comienzos de la web que estaba basado en el concepto de la herramienta hipertextual de software de Mac denominada HyperCard. Sin embargo, los investigadores generalmente están de acuerdo en que el punto de inflexión de la World Wide Web comenzó con la introducción del navegador web Mosaic, desarrollado por Marc Andreessen y sus compañeros en la Universidad de Illinois en Urbana-Champaign. *Mosaic* fue liberado en 1993, y a los pocos años se siguieron numerosas versiones comerciales, de las cuales la más usada fue *Netscape Navigator* e *Internet Explorer*. La razón básica del tremendo éxito de la web se puede resumir sucintamente: provee una forma adecuada de distribuir la información por Internet. Los individuos pueden publicar información y pueden acceder a otra por ellos mismos, sin necesitar aprendizaje ni ayuda.

3.3. Web semántica

La Web ha cambiado profundamente la forma en la que nos comunicamos, hacemos negocios y realizamos nuestro trabajo. La comunicación prácticamente con todo el mundo en cualquier momento y a bajo coste es posible hoy en día. Tenemos acceso a millones de recursos, independientemente

de nuestra situación geográfica e idioma. Sin embargo, al mismo tiempo, estos factores que han propiciado el éxito de la Web, también constituyen un desafío para las demandas cada vez mayores de los usuarios, dado el crecimiento exponencial de la cantidad de recursos accesibles y la heterogeneidad de estas fuentes de información con el consiguiente problema de interoperabilidad. Para ayudar a resolver estos invonvenientes se hace necesario un nuevo planteamiento en la organización y gestión de la información global.

El lenguaje HTML es válido para adecuar el aspecto visual del documento e incluir objetos multimedia en el texto (imágenes, esquemas de diálogo, etc.), pero da pocas posibilidades para categorizar los elementos que configuran el texto más allá de las típicas funciones estructurales, como sucede con otros lenguajes de maquetación (tipo LaTeX). Para dotar de significado los documentos que conforman la Web se necesita por tanto de algún mecanismo que permita enriquecerlos con información acerca de su contenido o dicho de otro modo **metadatos**. Este mecanismo permitiría una conexión semántica entre los recursos de la Web, que a su vez produciría una gestión ”inteligente” de los recursos mejorando las búsquedas de información, extracción de contenidos y la integración de recursos heterogéneos. Dicho de otro modo lo que se busca es la interoperabilidad semántica. Como se propone en el Marco Europeo de Interoperabilidad, ésta contempla no sólo que los recursos de información puedan estar conectados, sino que también la información pueda ser interpretable de forma automática y consecuentemente reutilizable por aplicaciones informáticas que no intervinieron en su creación. Al fin y al cabo todo esto se resume en conseguir tener una Web más útil y más inteligente que convierta la información en conocimiento mediante la incorporación de registros de metadatos.

La Web Semántica se ocuparía de resolver estas deficiencias. Para ello dispone de tecnologías de descripción de los contenidos, como RDF y OWL, además de XML, el lenguaje de marcas diseñado para describir los datos. Estas tecnologías se combinan para aportar descripciones explícitas de los recursos de la Web (ya sean éstos catálogos, formularios, mapas u otro tipo de objeto documental). De esta forma el contenido queda desvelado, como los datos de una base de datos accesibles por Web, o las etiquetas inmersas en el documento (normalmente en XHTML, o directamente en XML, y las instrucciones de visualización definidas en una hoja de estilos aparte).

Tim Berners Leen ya apuntaba desde los orígenes de la Web intenciones de añadir información descriptiva acerca de los recursos. Sin embargo, esto no fue posible al principio por diversos motivos. Por ejemplo, una de las ideas originales en su diseño era el ”tipado” de los enlaces, consistente en expresar la relación entre el sitio que enlaza a un segundo o más bien, la relación que mantienen las personas de ambos sitios (personal, profesional, etc). Esta idea podría aprovecharse por ejemplo para diferenciar en la página de un profesor los enlaces a las asignaturas que imparte de los enlaces de sus publicaciones.

El proyecto de la Web Semántica ha sido impulsado por el World Wide Web Consortium (W3C), dentro de su dominio de actividades denominado Technology & Society. Puede considerarse, además, un proyecto personal del inventor de la Web y fundador y director del W3C, Tim Berners Lee.

3.4. Conceptos básicos de la Web Semántica

Las tecnologías implicadas en la arquitectura y visualización de la Web semántica son múltiples y abarcan diversos campos y disciplinas. Aún así, en este apartado se apuntarán algunas de ellas que

suelen ser comunes y frecuentemente mencionadas:

3.4.1. Indexado y Recuperación de la información

Encontrar información cuando la búsqueda es compleja se vuelve, en ocasiones, una tarea ardua. Y esto aplica a cualquier medio: las bibliotecas tienen tarjetas del catálogo o índices electrónicos. Los motores de búsqueda son componentes vitales de la Web. Todavía hasta cierto punto, cualquiera se han frustrado por lo difícil que es encontrar algunas informaciones, especialmente cuando uno no está seguro sobre cómo preguntarlo. Para encontrar información, el enfoque de la Web Semántica espera ir más allá de los índices alfabéticos o por palabras clave para permitir realizar una búsqueda por conceptos y categorías. En este sentido, se necesita que los documentos sean capaces de declarar sus propios vocabularios y conjuntos de conceptos e identificar dónde son usados.

3.4.2. Metadatos

Un registro de metadatos consiste en un conjunto de atributos, o elementos, necesario para describir la fuente en cuestión. Por ejemplo, un sistema de metadatos común entre los bibliotecarios (el catálogo de biblioteca) contiene un conjunto de registros de metadatos con elementos que describen un libro u otra publicación en una biblioteca: autor, titulo, fecha de creación o publicación, materia, y la signatura topográfica especificando la localización de la publicación en el estante. A pesar de que el concepto de metadatos antecede a Internet y a la Web, el interés mundial por las normas y prácticas de metadatos ha estallado con el crecimiento de la publicación electrónica y las bibliotecas digitales, y la concurrente ”sobrecarga de información” que resulta de las grandes cantidades de datos digitales disponibles en línea. Cualquiera que pretenda encontrar información en línea utilizando cualquiera de los motores de búsqueda Web utilizados hoy en día, probablemente habrá experimentado la frustración al encontrar cientos, o miles, de resultados que satisfacen sólo parcialmente la búsqueda solicitada.

El término ”metadatos” no presenta una definición única. Según la definición más difundida, metadatos son ”datos sobre datos”. También hay muchas declaraciones como ”informaciones sobre datos”, ”datos sobre informaciones” e ”informaciones sobre informaciones”. Otra clase de definiciones trata de precisar el término como ”descripciones estructuradas y opcionales que están disponibles de forma pública para ayudar a localizar objetos” o ”datos estructurados y codificadas que describen características de instancias conteniendo informaciones para ayudar a identificar, descubrir, valorar y administrar las instancias descritas”.

La mayoría de las veces no es posible diferenciar entre datos y metadatos. Por ejemplo, un poema es un grupo de datos, pero también puede ser un grupo de metadatos si está adjuntado a una canción que lo usa como texto. Muchas veces, los datos son tanto ”datos” como ”metadatos”. Por ejemplo, el título de un texto es parte del texto como a la vez es un dato referente al texto (dato como metadato). Debido a que los metadatos son datos en sí mismos, es posible crear metadatos sobre metadatos, o metametadatos, datos que describan estos metadatos; el número de niveles de metadatos dependerá de la aplicación concreta o la disciplina que estemos tratando.

La adopción a gran escala de estándares y prácticas descriptivas para los recursos electrónicos mejorará la recuperación de recursos relevantes en cualquier contexto donde la recuperación es críti-

ca. Como señalan Weibel y Lagonze, dos líderes en el campo del desarrollo de metadatos:

> "La asociación de metadatos descriptivos normalizados a los objetos de la red tiene el potencial para mejorar sustancialmente las capacidades de localización/recuperación, facilitando búsquedas basadas en campos (p. ej. autor, título), permitiendo la indexación de objetos no textuales, y facilitando el acceso al contenido sustituido/referenciado que es distinto del acceso al contenido del propio recurso".

3.4.3. Anotaciones

En los libros ”materiales”, las personas solemos escribir notas al margen y comentarios, subrayamos y remarcamos pasajes, añadimos nuevos elementos, ideas y conceptos a los originales dados por el autor original. El estilo de los *wikis* permiten comentar y modificar las páginas web, pero este proceso sólo cubre una pequeña parte de lo que a la gente le gustaría hacer; sería deseable contar con mecanismos de anotación semántica, que permitisen realizar una correspondencia entre los conceptos presentados en un sitio web y las ontologías.

3.4.4. Una base datos interoperable

Hoy en día es muy común obtener la información desde una base de datos a lo largo de la Web. Estas bases de datos están separadas normalmente y no se usan fácilmente como fuentes de datos combinadas con otras bases, por lo que normalmente hay que desarrollar sistemas ad-hoc para resolver cada problema determinado. La Web Semántica da una visión en la que se unifique la descripción y recuperación de los datos almacenados, permitiendo que gran parte de la Web se considere como una gran base de datos virtual.

3.4.5. Recuperación automática de datos

Esta parte de la visión de la Web Semántica se centra en la adquisición automática de datos. Esto significa que un componente software con este objetivo , determina qué datos necesita, dónde y cómo consguirlo, para ir y obtener los datos.

3.4.6. Servicios

Un *servicio* es un comportamiento que provee un beneficio. Ejemplos de servicios incluyen tareas como hacer reservas, planificar horarios, dar precios, etc. Pensemos por ejemplo en un elemento perecedero como las flores o los alimentos. Una vez que se ha seleccionado el producto a comprar, hay que asegurarse de que su reparto cuadre con nuestro horario. El precio, las condiciones de compra, las opciones de reparto y nuestro propio horario se pueden considerar como servicios que deben ser activados y coordinados. Desde esta perspectiva de la Web Semántica, estos servicios publicarían información procesable automáticamente para llevar a cabo la activación y coordinación por nosotros mismos.

3.4.7. Descubrimiento

Para usar servicios, una persona y especialmente el software deben ser capaces de encontrarlos, descubrir lo que hacen y aprender a invocarlos. La aproximación más obvia consistiría en crear directorios de servcios con métodos de acceso estándar. Los servicios se describirían en términos estándar, y la información sobre cómo acceder a ellos también se codificaría de una forma normalizada.

3.4.8. Agentes inteligentes

Un *agente* es algo o alguien que actúa en nuestro nombre. Un agente software actuaría de una manera autónoma, comunicándose con otros agentes software (que pueden ser más especializados) para encontrar servicios, productos o información. Una red de estos agentes que interactúan entre sí sería capaz de describir sus objetivos usando vocabularios establecidos, para descubrir servicios e recursos de información, y para usar la mayor parte de las capacidades descritas en los puntos anteriores.

Por ejemplo, uno de estos agentes especializados podría ”saber” cómo comprar un billete de avión y hacer reservas. Otro agente podría llevar a cabo los servicios necesarios, devolviendo los resultados a nuestro agente, que a su vez nos lo notificaría.

3.5. Capas de la Web Semántica

El W3C, con Tim Berners-Lee al frente, ha sido (y sigue siendo) un líder en desarrollo de tecnologías para la Web, como XML o RDF. Dentro del sitio web del W3C podemos encontrar páginas dedicadas a la Web Semántica, en las que se incluye un diagrama etiquetado como *”Semantic Web layer cake”* o tarta de la Web Semántica.

En él se pueden encontrar las siguientes capas:

- Unicode: la codificación de todos los símbolos usados en los distintos idiomas para que cualquier texto se pueda expresar en la Web.
- URI (*Uniform Resource Identifier*): una ampliación de las actuales direcciones que se forma combinando las URL con las URN. Así, mientras las URL permiten la localización de las webs, las URN describen los recursos del dominio.
- XML (*Extensible Markup Language*): el framework de lenguages que, desde 1998, ha sido utilizado para definir muchos de los nuevos lenguajes que se usan actualmente para intercambiar datos en la Web.
- XML Schema: un lenguaje usado para definir la estructura específica de los lenguajes basados en XML.
- RDF (*Resource Description Framework*): un lenguaje flexible capaz de describir todos los tipos de información y metadatos.

- RDF Schema: una plataforma que provee los medios para especificar el vocabulario básico que usarán específicamente los lenguajes de aplicación basados en RDF.
- Ontologías: lenguajes utilizados para definir vocabularios y establecer las guías de uso de las palabras y términos en el contexto de un vocabulario específico. RDF Schema es un framework para la construcción de ontologías. OWL es un lenguaje de ontologías diseñado para la Web Semántica.
- Lógica: el razonamiento lógico es empleado para determinar la consistencia y corrección de los conjuntos de datos y para inferir nuevas conclusiones que estuvieran explícitamente dadas de partida.
- Pruebas: llevan a cabo el seguimiento o explican los pasos dados en el razonamiento lógico.
- Confianza: un medio de dar autenticación de identidad y evidencia de la confianza de los datos, servicios y agentes. Los agentes serán escépticos sobre lo que leen en la web semántica hasta que hayan comprobado de forma exhaustiva las fuentes de información.
- Firma digital: contiene el conjunto de datos encriptados que utilizan los ordenadores y los agentes para verificar que la información pertenece a una web específica y fiable.

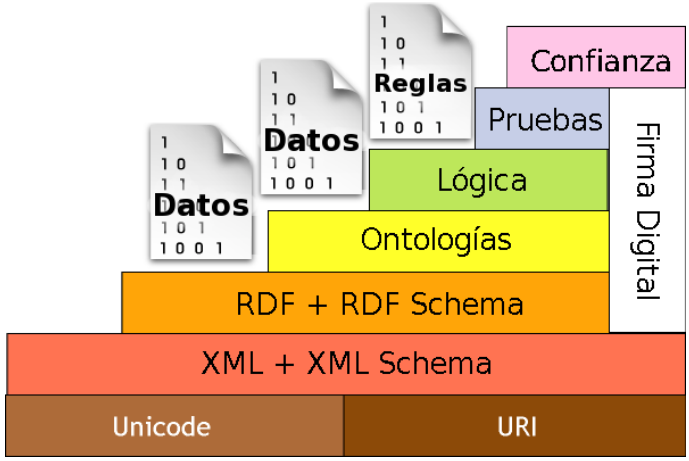


Figura 3.1: ”Tarta” de la Web Semántica

3.6. Web como Biblioteca digital

Las capacidades que ofrecen estas tecnologías para compartir y reutilizar recursos tienen entre sus diversas aplicaciones el campo de las bibliotecas digitales. No es nada fácil en estos momentos definir qué es una biblioteca digital, es decir, lo que cada uno de nosotros entiende por biblioteca digital, y más concretamente, distinguir entre el uso y el significado de los diferentes términos que se utilizan para definir esta realidad.

Básicamente, se han utilizado y mezclado tres conceptos que, pese a tener connotaciones diferentes, muchas veces han pretendido definir lo mismo: biblioteca electrónica, biblioteca digital y biblioteca virtual.

Varias son las definiciones que se han aplicado a las bibliotecas digitales. Algunas defienden que las bibliotecas digitales son meramente bibliotecas electrónicas. La biblioteca electrónica sería aquella que permite acceder a bancos de información en formato electrónico. Este tipo de bibliotecas incluiría también los catálogos automatizados de bibliotecas tradicionales. Según esta definición. la biblioteca electrónica intentaría reproducir la producción impresa pero utilizando un medio diferente del soporte papel. Partiendo de esta realidad la biblioteca digital seguiría los pasos de la biblioteca electrónica, pero evolucionando hacia la introducción de otros tipos de materiales, es decir, introduciendo elementos digitales.

Otras definiciones proponen un enfoque más tecnológico, e incluyen servicios que se ofrecen aprovechando los sistemas de distribución de las redes, los cuales permiten acceder a dichos servicios desde cualquier lugar, a cualquier hora, cualquiera persona e incluso, en algunos casos, sin gastos.

En la web del Digital Library Project, hay una definición de biblioteca digital, que proviene del Santa Fe Workshop on Distributed Knowledge Work Environments [DA97] y que en opinión de esta misma web es una de las mejores definiciones. Dice así: "El concepto de biblioteca digital no es únicamente el equivalente de repertorios digitalizados con métodos de gestión de la información. Es más bien, un entorno donde se reúnen colecciones, servicios, y personal que favorece el ciclo completo de la creación, difusión, uso y preservación de los datos, para la información y el conocimiento".

La mayoría de los expertos en biblioteconomía y documentación definen las bibliotecas digitales como repertorios de objetos digitales, más o menos organizados, que sirven a una comunidad de usuarios definida, los cuales tienen los derechos de autor presentes y gestionados, y disponen de mecanismos de preservación y conservación. Esta definición tiene en cuenta que estos repertorios constan de datos (el contenido) y metadatos (la información que describe los datos) e incorporan técnicas de búsqueda y recuperación de la información.

Hay otras definiciones que hacen hincapié tanto en la interacción de los ordenadores y las personas como en las interfaces que permiten acceder a la información mediante ciertos mecanismos: búsqueda, navegación, enlaces hipertextuales, etc. Asimismo, estas definiciones enfatizan que en estas bibliotecas se tratan los datos teniendo en cuenta el ciclo de la gestión del conocimiento: organización, comunicación/difusión, almacenaje, búsqueda, filtrado/selección, y reutilización.

Por lo general, las bibliotecas digitales son implementadas por instituciones culturales cuyo objetivo es hacer accesibles sus fondos a los usuarios.

El concepto de biblioteca digital lleva implícito un proceso de innovación tecnológica que modifica la producción, la organización y la difusión de la información.

Las bibliotecas digitales incluyen una enorme gama de tipologías. No ofrecen únicamente producción impresa, sino que incluyen imágenes, vídeos, sonido, reproducción de elementos en 3D, datos, mapas, etc. Los campos que cubren son multidisciplinares y van desde la literatura y el arte hasta la música, la medicina, etc..

La biblioteca digital no intenta ”copiar” la realidad impresa , sino que genera una nueva estructura de la información que hace que ésta evolucione desde el concepto lineal del libro y los documentos

tradicionales al concepto hipertextual, donde la información llega al usuario de formas muy variadas y provista de todo tipo de vínculos, los cuales permiten ampliar, concretar o explicar los contenidos de forma simultánea y diferente. El hipertexto incluye mucha más información no textual que el impreso, ya que incorpora elementos multidimensionales: voz, sonido, imagen, 3D, etc..

Con todas estas definiciones podríamos hacer el siguiente esquema:

- Biblioteca clásica: contenidos en soportes físicos, acceso mediante referencias bibliográficas consignadas en los catálogos.
- Biblioteca electrónica: contenidos en soporte electrónico, acceso por medios físicos (CD-ROM), o electrónicos (acceso en línea).
- Biblioteca digital: contenidos en soportes electrónicos y digitales, y acceso en línea a través de redes telemáticas.
- Biblioteca virtual: contenidos en soporte electrónico y digital, y acceso en línea a través de redes telemáticas (como en las bibliotecas digitales).

Sin embargo, no todo es fácil ni simple a la hora de pensar en la biblioteca digital, existen una serie de problemáticas que ponen freno su rápida expansión, mencionaremos algunas de ellas:

- Disponibilidad: todo lo que existe registrado (impreso, fotografiado, filmado, pintado, dibujado, etc.) tendría que convertirse a formato digital para que éste disponible a todos los usuarios con un terminal de trabajo.
- Recuperación y adecuación: cada usuario de este hipotético terminal de trabajo (que permitiría el acceso a la biblioteca digital) tendría que poder acceder a todos los documentos electrónicos relevantes de este universo digital, de una manera rápida y fácil.
- Autenticidad: cada usuario debería tener la seguridad de que el documento que encuentra en la red es el documento auténtico y original.
- Utilización: cada uno de los documentos recuperados mediante el terminal de trabajo tendría que ser recuperado de forma que todo usuario pudiera utilizarlo.
- Protección de la propiedad intelectual: la protección de los derechos de autor debería estar garantizada en todo documento recuperado. La propiedad intelectual de un recurso es un tema candente actualmente en sí mismo y que incluyen líneas de discusión desde las licencias o condiciones bajo las cuales se permite el uso, copia, distribución, etc. hasta el sujeto mismo al que afecta la propiedad intelectual (la obra en sí, el objeto digitializado, cada una de las instancias digitales del recurso, etc).
- Asequibilidad: los costes de acceso y recuperación de los diversos documentos tendrían que ser razonables y no superar los costes de sus equivalentes tradicionales.

La ARL (*Association of Research Libraries*) señala unos elementos comunes a los diversos términos con los que se designan las bibliotecas digitales (bibliotecas electrónicas, bibliotecas virtuales, etc.). Algunos de estos elementos son:

- La biblioteca digital no debe ser una entidad individual.
- La biblioteca digital requiere que haya medios tecnológicos para enlazar recursos.
- Los enlaces entre un gran número de bibliotecas digitales y los servicios de información deben ser transparentes para los usuarios. El acceso universal a las bibliotecas digitales y a los servicios de información debe ser un objetivo principal.
- Las bibliotecas digitales no deben limitarse a suplir documentos, sino que deben ofrecer otros elementos digitales que no pueden suministrarse en formato impreso.

Una de las características de las bibliotecas digitales es que la información que contienen ha sido creada por gente diversa, utilizando medios diversos, dándole formas y formatos diferentes, almacenada en diferentes lugares del mundo (servidores) y de manera creciente e interconectada por medio de redes. Es decir, en estas bibliotecas conviven materiales en diferentes formatos, en distintas versiones, ubicados en diferentes lugares, y accesibles a un gran número y diversidad de personas.

Los proyectos de bibliotecas digitales y la investigación en estos temas deben permitir el cambio continuo, debido al aumento del ancho de banda de las redes de comunicaciones, las cuales permiten gestionar y dar coherencia, utilizar y posibilitan el acceso a gran cantidad de datos distribuidos y transformados en información y conocimiento.

La existencia de las bibliotecas digitales hace cada vez más necesario que haya sistemas de recuperación de la información que sean capaces de procesar el lenguaje natural. Estos sistemas recuperan y seleccionan frases lingüísticas como unidades de información y además recuperan y seleccionan términos controlados que forman parte de tesauro, o términos incluidos en una estructura de árbol del conocimiento.

Estos sistemas de recuperación tienen que ser:

- Flexibles: capaces de procesar diferentes tipos de información
- Precisos: capaces de seleccionar información pertinente y desestimar el ruido".
- Rápidos: tiene que poder tratar simultáneamente cantidades ingentes de información y documentación
- Automáticos: capaces de seleccionar la información sin que tenga que estructurarse antes
- Fáciles: su utilización no tiene que suponer un problema para el usuario

Paralelamente al gran desarrollo de las bibliotecas digitales ha surgido la necesidad de procesar los contenidos de estos repertorios para facilitar la búsqueda y la recuperación de la información de una forma eficaz. La biblioteca digital tiene que cumplir una serie de características que le den el valor que necesita para difundir estos contenidos. Tienen que ser recuperables mediante metadatos que proporcionen valor añadido a la mera acumulación de información. Los metadatos tienen una gran importancia en la composición de las bibliotecas digitales, ya que permiten una búsqueda efectiva y precisa.

3.7. Metadatos y Bibliotecas Digitales

De la misma forma que en general la asignación de metadatos a los contenidos de la Web es fundamental para un mejor aprovechamiento de la misma, particularmente, las bibliotecas digitales componen un vasto campo en el que la aplicación de estándares de metadatos es clave para la gestión y recuperación de sus recursos. Entre los beneficios potenciales de las bibliotecas digitales, encontramos que la biblioteca digital trae la biblioteca al usuario, se aprovecha la potencia del computador para la búsqueda, la información puede compartirse, es más fácil mantener la información actualizada, no atiende a horarios y siempre está disponible, permite nuevas formas de distribución y suele tener un coste menor que el equivalente de una biblioteca tradicional.

Casi cualquier biblioteca tiene un catálogo con registros de los materiales que componen sus colecciones. El catálogo ayuda a los usuarios a encontrar materiales en la biblioteca, provee información bibliográfica, y además es una herramienta considerable a la hora de mantener y gestionar las colecciones. El catalogado es un área en el que las bibliotecas usan una terminología precisa, algunas de las cuales pueden resultar desconocidas para personas que no pertenecen al campo. Mientras la palabra catálogo puede resultar un término genérico, en el contexto de una biblioteca tiene un significado muy concreto: una colección de registros bibliográficos creados según una serie de reglas estrictas. Más que la palabra libro, los bibliotecarios suelen usar el término monografía. Así, a lo largo de los años, la información de un registro del catálogo de monografías ha sido codificado mediante reglas de catalogado, como por ejemplo AACR (*Anglo-American Cataloging Rules*) en los países de habla inglesa.

La tarea de catalogar cada monografía suele costar un tiempo considerable y requiere bastante experiencia. Para ahorrar costes y no duplicar información, las bibliotecas comparten sus registros de catálogos. Así se tiene que la Biblioteca del Congreso y las mayores bibliotecas de universidades disponen sus catálogos entre sí sin coste alguno.

Los primeros intentos de almacenar información bibliotecaria en computadores, a finales de 1960, se encontró con serias barreras técnicas, entre las que se incluyen el alto costo de los propios computadores, interfaces de usuarios rígidas y la carencia de redes para el intercambio. Como el almacenamiento era costoso, las primeras aplicaciones tuvieron lugar en áreas donde los beneficios financieros pudieran ser más rentables que almacenar la información en volúmenes de bibliotecas tradicionales.

Uno de los primeros éxitos fue el desarrollo de la Biblioteca del Congreso llamado MARC, un formato para el catalogado legible por máquinas (*MAchine-Readable Cataloging*). El uso de MARC por parte del OCLC (*Online Computer Library Center*) para compartir registros de catálogo entre numerosas bibliotecas resultó en importantes ahorros para las bibliotecas. MARC fue desarrollado por Henriette Avram y sus colegas en la Biblioteca del Congreso, inicialmente como un formato para distribuir registros del catálogo en cintas magnéticas. En la práctica, el término de catalogado MARC se usa a menudo con un sentido general para cubrir tanto los registros MARC como el formato electrónico con el que se almacenan. El desarrollo de MARC llevó a dos importantes tipos de sistemas computacionales. El primero fue el catalogado compartido, con Fred Kilgour como pionero, y fundador del OCLC en 1967. EL OCLC tiene un gran sistema de computadores con varias decenas de millones de registros de catálogo en formato MARC, de forma que cualquier miembro del mismo puede ir registrando nuevas monografías de forma que cada item se almacena una única vez, y así el esfuerzo intelectual se comparte entre las distintas bibliotecas. Por otra parte, la disponibilidad de

los registros MARC estimuló un segundo desarrollo: las bibliotecas empezaron a crear catálogos online de forma individual (OPAC - *online public access catalog*). Para conectar estos catálogos online que empezaban a emerger a finales de los 70, comenzó un proyecto conocido como *Linked Systems Project*, que desarrolló un protocolo conocido ahora por el nombre Z39.50. Este protocolo permite a una computadora buscar información en otras. Primariamente se usa como medio para buscar en registros MARC, pero el protocolo es flexible y no está restringido a MARC. Técnicamente, el protocolo Z39.50 especifica un conjunto de reglas que permiten a una computadora buscar en la base de datos de otra y recuperar los registros que encuentre. En la figura 3.2 puede verse la distribución de servidores Z39.40 en España.

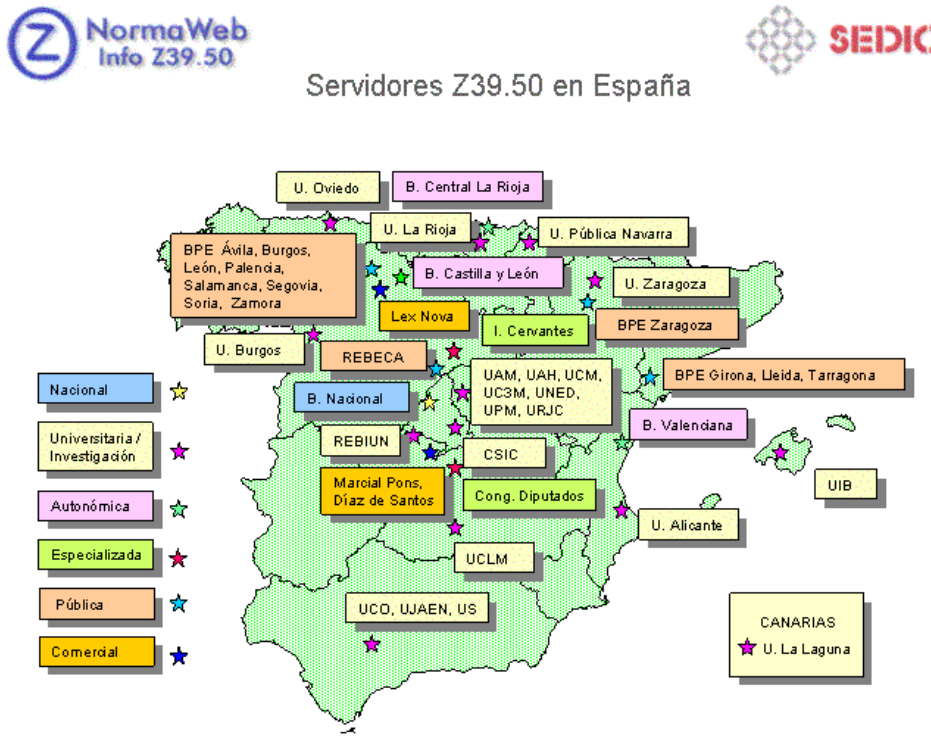


Figura 3.2: Servidores Z39.50 en España

MARC fue un formato realmente innovador en un tiempo en el que la mayoría de los sistemas de computadores representaba el texto como campos de longitud fija en mayúsculas exclusivamente. Todavía se mantiene hoy en día como un formato vital para las bibliotecas, pero ya se va notando su edad. Sin embargo, cualquiera que sea su futuro, MARC representó el logro pionero tanto en la historia de los computadores como en la de las bibliotecas. De ahí que que este formato haya evolucionado hacia el MARC 2.0, que es MARC21. En los países europeos Marc es algo innegociable, porque está totalmente aceptado. Ahora se deben construir las interacciones entre la biblioteca convencional y la biblioteca digital y lo ideal es conciliar estas cuestiones.

Desde el punto de vista de organismos federales, las bibliotecas digitales no fueron una materia

explícita de investigación hasta los 90. En 1992, DARPA fundó el proyecto *Computer Science Technical Reports*, que estaba coordinado por la CNRI (*Corporation for National Research Iniatives*), y que involucraba a cinco universidades: Carnegie Mellon, Cornell, MIT, Stanford y la Universidad de California en Berkeley. A pesar de todo, la iniciativa que realmente estableció a las bibliotecas digitales como un campo distinto de investigación se dio en 1994, cuando las divisiones de ciencias de la computación de la NSF, DARPA y NASA crearon la Iniciativa de Bibliotecas digitales (DLI - *Digital Libraries Iniative*). Se hizo inversión para seis proyectos de cuatro años con el objetivo de implementar un banco de pruebas sobre bibliotecas digitales:

- La Universidad de California en Berkeley construyó una gran colección de documentos del entorno de California, incluyendo mapas, imágenes e informes del gobierno. Además, se hizo una notable investigación y trabajo con documentos multivalentes, Chesire II (un sistema de búsqueda que combinaba la potencia de los formatos SGML con la información de registros MARC), y reconocimiento de imágenes.
- La Universidad de California en Santa Bárbara se concentró en mapas y otros tipos de información geoespacial. Su colección se llama *Alexandria Digital Library*. En esta línea su trabajo consistió en la inclusión de metadatos para información geoespacial, wavelets para la compresión y transmisión de imágenes, y nuevos métodos para analizar cómo usa la gente las bibliotecas.
- La Universidad Carnegie Mellon construyó una biblioteca de segmentos de vídeo, llamada Informedia. Este trabajo hizo énfasis en el procesado automático para el descubrimiento de información, búsqueda multimodal, reconcimiento de voz, reconocimiento de imágenes y *skimming* de vídeo.
- La Universidad de Illinois trabajó con publicadores para construir una biblioteca federal de periódicos y revistas de ciencia e ingeniería. La mayor parte de su trabajo se concentró en la manipulación de documentos SGML. Además, este proyecto también usaba supercomputación para estudiar los problemas de la información semántica en colecciones muy grandes de documentos.
- La Universidad de Michigan desarrolló un proyecto sobre las colecciones de las bibliotecas digitales de las universidades. Además, los investigadores llevaron a cabo diversos experimentos con modelos económicos y con un enfoque basado en agentes para la intereoperabilidad.
- La Universidad de Stanford se centró en un proyecto llamado InfoBus, que se basaba en un método de combinar servicios de diversos orígenes en un conjunto coherente de servicios de bibliotecas digitales.

La Iniciativa de Bibliotecas Digitales más allá del trabajo específico que propició, dió forma a una disciplina emergente. La investigación en este campo no era un tema nuevo, pero hasta entonces se había llevado a cabo de una forma fragmentada. Incluso el nombre ”Biblioteca digital” era un término incierto. El establecimiento de este nuevo campo fue importante ya que creó la suficiente confianza necesaria para que el trabajo tanto en investigacación como en aplicaciones prácticas continuara a largo plazo.

Además, la Iniciativa de Bibliotecas digital aclaró la distinción entre investigación e implementación. Los proyectos que estaban proponiendo eran trabajos de verdadera investigación; algunos de

ellos ya se han migrado a aplicaciones prácticas; otros anticipan desarrollos software y hardware, mientras que otros son meramente experimentales. La emergencia de las bibliotecas digitales como una disciplina de investigación podría correr el riesgo de que los investigadores se centraran en problemas demasiado teóricos, pero se contaba con que los fundadores de esta iniciativa eran tres organizaciones federales. De esta manera, la primera fase de la investigacación enfatizó sobre todo en los aspectos pertinentes de ciencias de la computación relativos a este nuevo campo. Sin embargo, las agencias fundadoras (DARPA, NASA y NSF) sabían que esta disciplina era más que una rama de las ciencias de computación. En 1998, cuando se pasó a la segunda fase, incorporó otros grupos de trabajo provenientes de otras disciplinas. Con ello, la orientación del programa se mantenía eminentemente práctica y aplicada, ya que los criterios de concesión de ayudas potenciaron los proyectos de colaboración que presentaban un uso innovador de la tecnología y que tuvieran un carácter práctico con mucha proximidad al mercado.

3.8. Recursos en las Bibliotecas Digitales

Dentro del concepto de biblioteca digital un concepto importante es cómo entender los objetos que la conforman. Las bibliotecas digitales pueden contener tipo de información que se pueda codificar como secuencias de bits, o llamados de forma general, recursos. Según la definición que da W3C/IETF un recurso es cualquier cosa que tenga una identidad. Ejemplos cotidianos de recursos incluyen un documento electrónico, una imagen, un servicio de partes climatológicos, y una colección de otros recursos. No todos los recursos son recuperables via red, como son las personas, empresas, libros ”físicos” sujetos a una biblioteca, etc. Esta definición nos lleva a considerar un recurso como cualquier cosa, ya tenga una naturaleza física (libros, coches, personas), digital (páginas web, obras electrónicas) o conceptual (colores, ideas).

La adopción de estándares es clave para poder compartir de una forma efectiva los recursos digitales y permitir una interoperabilidad entre instituciones. Ya durante la pasada década han venido emergiendo nuevos enfoques y estándares para la descripción de recursos digitales, como pueden ser, entre otros, *Machine Readable Cataloging* (MARC), *Anglo-American Cataloging Rules, second edition* (AACR2), *Visual Resources Association Core Schemas* (VRA), *Categories for the descriptions of works of art* (CDWA) o *Dublin Core* (DC), que veremos a continuación.

Estos esquemas de metadatos, junto a la codificación sintáctica XML/RDF, normas de descripción de contenido -ontologías, topic maps, tesauros, etc.- y toda una serie de protocolos para el intercambio de información, protagonizarán la segunda generación de la web, donde las bibliotecas digitales ganarán mayor terreno al brindar la información sistematizada y estructurada dentro del entramado complejísimo en que se nos presenta Internet.

3.9. Esquema de Metadatos para el proyecto Virtual Museum

Después de esta descripción a grandes rasgos de la web semántica y metadatos, parece razonable la necesidad de adopción de un estándar de metadatos para el caso particular de museos virtuales y por tanto del presente proyecto. Aquí, se hará uso de Dublin Core. Aunque se describirán las características que lo hacen apropiado en nuestro contexto, se pueden adelantar algunas como son:

- Versatilidad
- Independencia sintáctica
- Interoperabilidad semántica
- Simplicidad
- Normalización formal
- Evolución a través de una institución consorciada: DCMI
- Modularidad y arquitectura de metadatos

3.10. Origen de la Iniciativa Dublin Core

La Iniciativa de Metadatos Dublín Core (DCMI), (http://dublincore.org), es una organización cuyos fines son la promoción y difusión de normas interoperables sobre metadatos, así como el desarrollo de vocabularios especializados controlados para representar recursos que permitan el desarrollo de sistemas de recuperación más inteligentes.

Se podría decir que Dublin Core proviene de dos circunstancias principales:

- Por un lado, en un hecho real, una necesidad y una coyuntura informativa. En este caso, la imposibilidad de catalogar la Web a través del formato MARC, que habían evidenciado, ya en 1995, proyectos como Intercat de OCLC12 o CATRIONA en el Reino Unido.
- Y por otro, un cúmulo de casualidades y buenas intenciones de un grupo humano.

Dublin Core tiene sus orígenes en Chicago durante la segunda Conferencia Internacional World Wide Web, en octubre de 1994. Yuri Rubinsky de SoftQuad junto a Stuart Weibel y Eric Miller de OCLC, tuvieron una conversación con Joe Hardin, director del Centro Nacional para Aplicaciones de Supercomputación (*National Center for Supercomputing Applications* - NCSA) que les llevó a una discusión sobre semántica y la web.

La confrontación inicial de ideas llevó al NCSA y OCLC a formar un taller de trabajo para discutir la semántica de los metadatos en Dublín, Ohio, en marzo de 1995. En este evento, llamado simplemente "Taller de Metadatos OCLC/NCSA", más de 50 personas discutieron cómo un conjunto de recursos semánticos serían muy útiles para facilitar la búsqueda y la recuperación en la web. Al resultado le denominaron "Metadatos Dublín Core", por el lugar donde se realizó el taller. Desde entonces, se han realizado un total de ocho talleres en Inglaterra, Australia, Finlandia, Alemania, Canadá y los Estados Unidos, a los que han seguido una serie de conferencias internacionales que de momento tienen una frecuencia anual lo que da una idea del estado actual de este proyecto.

La DCMI es la iniciativa internacional de metadatos más sólida e importante para la organización y recuperación de información en Internet en forma normalizada, eficaz y con un propósito general. Es hoy un esquema maduro de metainformación cuyo conjunto de elementos (DCMES - *Dublin Core Elements Metadata Set*) se ha formalizado, primero como norma ANSI/NISO Z39.85 en octubre de

2001, y posteriormente, como estándar internacional ISO 15836-2003, desde el 8 de abril de 2003, con independencia de que, en cada ámbito de información, se desarrollen esquemas propios de metainformación para mejorar la recuperación en la red. Este nivel de normalización formal constituye una de las razones de su éxito; uno de los problemas habituales de los estándares para la Web es que se desarrollan y utilizan en un nivel *de facto* o de especificaciones de dominio público, siendo muy pocos los que alcanzan en nivel de reconocimiento como estándar formal (*de iure*) ISO. Desde su proclamación como estándar ISO, distintos países han mostrado su credibilidad en este esquema de metadatos, reconociéndolo como estándar nacional, por ejemplo en España, se ha convertido en la norma UNE-ISO 15836:2007.

En general, las principales características de Dublin Core, y que lo hacen particularmente apropiado en este proyecto son:

- **Simplicidad:** La simplicidad reduce considerablemente los costos y promueve la interoperatividad. La simplicidad no significa acomodar o desechar la semántica y las funciones enriquecidas, soportadas por otros sistemas complejos de metadatos. De hecho, Dublín Core estimula el uso de formatos de metadatos enriquecidos en combinación con Dublín Core y, a menudo, es también, el punto de partida para la creación de descripciones más complejas.

- **Interoperatividad semántica:** Las diferencias en la terminología y en las prácticas descriptivas entre un campo del conocimiento y otro impiden la localización/recuperación de información a través de la amplitud del espacio de Internet. El Dublin Core puede ayudar al ”turista digital” –alguien no especializado que busca información– a encontrar su camino a través de un conjunto de elementos común, cuya semántica es universalmente entendida y soportada. Por ejemplo, los científicos preocupados por localizar artículos por un autor particular, y alumnos de arte interesados en trabajos de un artista particular, pueden estar de acuerdo en la importancia del elemento ”creator”. Tal convergencia en un conjunto de elementos común, a pesar de que sea ligeramente más genérica, aumenta la visibilidad y la accesibilidad de todos los recursos, tanto dentro de una disciplina determinada, como más allá de ésta.

- **Flexibilidad:** Nada en el Dublín Core es obligatorio, todos los elementos son opcionales y repetibles, así el usuario elige la profundidad de su descripción. El formato además, permite incorporar desde estructuras simples hasta aquellas más elaboradas semánticamente.

- **Extensibilidad:** Propiedad que deriva, en parte, de la flexibilidad y, en parte, de la definición de elementos estructurados con distintos grados de complejidad y requerimientos. También, se relaciona con su capacidad de convertirse fácilmente a otros formatos, entre ellos y a pesar de su complejidad, al propio formato MARC.

- **Alcance Internacional:** El Conjunto de Elementos Dublin Core se desarrolló originalmente en inglés, pero se han creado versiones en otras muchas lenguas, como por ejemplo finlandés, noruego, tailandés, japonés, francés, portugués, alemán, griego, indonesio y español. El Grupo de Interés especial en localización e internacionalización está coordinando esfuerzos para aunar estas versiones en un registro distribuido. Aunque los retos técnicos de internacionalización de la World Wide Web no se dirigen directamente por la comunidad de desarrollo del Dublin Core, la participación de representantes de prácticamente todos los continentes, ha asegurado que el desarrollo del estándar considere la naturaleza, multilingüe y multicultural, del universo de información electrónica.

El éxito del Dublin Core y de la utilización y adopción de sus elementos se debe a varias razones:

- Define una semántica precisa pero es independiente sintácticamente; es decir, no depende de una sintaxis de codificación particular, ni HTML, ni XML, ni RDF, sino todas ellas.
- Se ha adoptado internacionalmente y sus elementos y semántica asociada están traducidos a más de 20 idiomas.
- Es un estándar de propósito general, no depende de ningún dominio informativo, pero se adapta a las distintas comunidades de información Web y se lleva muy bien con otros esquemas de metadatos de propósito específico, sirviendo de piedra rosetta para la representación de las relaciones entre elementos.
- Es el modelo de metadatos clave en sistemas y servicios de información digital, como por ejemplo para la iniciativa de archivos abiertos (OAI), o servicios comerciales como Connexion de OCLC. También ha sido adoptado por distintos gobiernos en sus proyectos de e-Gov, por ejemplo en: Australia, Canadá, Dinamarca, Finlandia, Irlanda, Nueva Zelanda y Reino Unido), y por los más emblemáticos proyectos de bibliotecas digitales y gestión del patrimonio digital.
- Tiene una gran validez como estándar porque es simple, extensible e interoperable. El hecho de ser una norma ISO, lo convierte en un estándar apto para la industria y así lo demuestra su uso en contextos corporativos y el protagonismo que ha adquirido en algunos sistemas gestores de contenidos empresariales (ECMS) como *Libertas Solutions CMS* o *CONTENS*. Incluso está presente como *plugins* en sistemas de gestión de blogs como *WordPress* o navegadores como *Mozilla Firefox*.
- Voluntad de que una Web mejor es posible y al espíritu abierto, global e independiente de la DCMI que hace que sea una iniciativa viva que se adapta a las necesidades de distintos tipos de usuarios o distintos tipos de información, creando más términos de metadatos, más perfiles de aplicación, o simplemente adaptando el uso de los elementos a un fin particular.

3.11. Conjunto de Elementos de Metadatos

La estructura de Dublin Core está basada en dos niveles: simple y cualificado. El nivel simple está formado por 15 elementos, mientras que el nivel cualificado incluye dos elementos más así como un conjunto de nuevos elementos cualitativos llamados cualificadores, dedicados a la descripción más en detalle de los elementos simples. Se puede ver a los elementos como nombres, y a los elementos cualitativos como adjetivos, cuya misión es concretar más el significado del nombre, pero nunca extenderlo. Además, los elementos cualitativos deben cumplir el principio de mutismo (Dumb-Down), en virtud del cual los elementos cualitativos pueden llegar a ser mudos; es decir, todo elemento debe ser entendido sin necesidad de los elementos cualitativos, de tal forma que un usuario siempre podrá usar un metadato sin necesidad de ellos. Finalmente, cada elemento es opcional y se puede repetir.

El estándar de Dublin Core es un conjunto de elementos, sencillo pero efectivo, para describir un amplio abanico de recursos en red. El estándar de Dublin Core consta de 15 elementos, cuya semántica ha sido establecida mediante consenso por un grupo internacional de profesionales de

distintas disciplinas, como biblioteconomía, informática, codificación de textos, museología y otros campos de conocimiento relacionados.

$$Contenido \begin{cases} Coverage \\ Description \\ Type \\ Relation \\ Source \\ Subject \\ Title \end{cases} \quad PropiedadIntelectual \begin{cases} Contributor \\ Creator \\ Publisher \\ Rights \end{cases} \quad Aplicacion \begin{cases} Date \\ Format \\ Identifier \\ Language \end{cases} \tag{3.1}$$

3.11.1. Descripción de los elementos

Title (Título)

El título dado al recurso por su creador o editor

Creator (Creador)

La persona u organización responsable del contenido intelectual del recurso

Subject (Claves)

El tema del recurso: palabras clave o frases que describen el tema o el contenido del recurso. Se debería fomentar el uso de vocabularios controlados y de sistemas de clasificación formales.

Description (Descripción)

La descripción textual del recurso, tal como un resumen en el caso de un documento o una descripción del contenido en el caso de un documento visual.

Publisher (Editor)

La entidad responsable de que el recurso esté disponible en su forma presente, por ejemplo la empresa editora, un departamento universitario u otro tipo de organización.

Contributor (Otros colaboradores)

Persona u organización, en adición a aquellas especificadas en el elemento Creator, que han realizado contribuciones intelectuales significativas al recurso, pero cuya contribución es secundaria respecto a los individuos o entidades especificadas en el elemento Creator (por ejemplo, editores, traductores e ilustradores).

Date (Fecha)

La fecha en la que el recurso se puso a disposición del usuario en su forma actual. Esta fecha no ha de confundirse con la que pertenece al elemento Cobertura, que sería asociada con el recurso sólo en la medida en que el contenido intelectual está de algún modo relacionado con esa fecha.

Type (Tipo del Recurso)

La categoría del recurso, tal como sede web, novela, poema, informe, informe técnico, ensayo, diccionario, etc.

Format (Formato)

La representación de datos del recurso, tal como text/html, ASCII, Postcript, aplicación ejecutable o imagen JPEG. Es usado para identificar el software y posiblemente, el hardware que se necesitaría para mostrar el recurso.

Identifier (Identificador del Recurso)

secuencia de carácteres utilizados para identificar unívocamente un recurso. Ejemplos para recursos en línea pueden ser URLs i URNs. Para otros recursos pueden ser usados otros formatos de identificadores, como por ejemplo ISBN (*International Standard Book Number*).

Source (Fuente)

El trabajo, ya sea impreso o electrónico, del que deriva el recurso, si es aplicable. Consta de una secuencia de caracteres utilizado para identificar unívocamente un trabajo a partir del cual proviene el recurso actual.

Language (Lengua)

Lengua(s) del contenido intelectual del recurso.

Relation (Relación)

Un identificador de un segundo recurso y su relación con el recurso actual. Este elemento permite enlazar los recursos relacionados y las descripciones de los recursos.

Coverage (Cobertura)

La característica de cobertura espacial y/o temporal del contenido intelectual del recurso. La cobertura espacial se refiere a una región, uso de coordenadas o nombres de lugares extraídos de una lista controlada. La cobertura temporal se refiere al contenido del recurso en vez de a cuando fue creado o puesto accesible ya que este último pertenece al elemento fecha.

Rights (Derechos)

Una referencia (URL, por ejemplo) para una nota sobre derechos de autor, para un servicio de gestión de derechos o para un servicio que dará información sobre términos y condiciones de acceso a un recurso.

3.11.2. Elementos cualitativos

Dublin Core reconoce actualmente dos grandes clases de cualificadores:

- **Refinamiento de elementos(*Element Refinement*):** Estos cualificadores refinan el significado de un elemento más o lo hacen más específico. Un elemento refinado comparte el significado del elemento sin cualificar, pero con ámbito más restringido. En cualquier caso, un elemento refinado debería poder comprenderse ignorando el cualificador y tratar el valor del metadato como si estuviese sin cualificar.
- **Esquema de codificación (*Encoding Scheme*):** Estos cualificadores identifican esquemas que ayudan a la interpretación del valor del elemento. Estos esquemas incluyen por ejemplo vocabularios controlados, notaciones formales o reglas de análisis. Un valor expresado que haga uso de un esquema de codificación será por tanto un elemento particular del vocabulario controlado, una cadena formateada de acuerdo a una notación formal, etc. Si el esquema de codificación no es comprensible por el cliente o el agente, el valor debería permanecer siendo útil para el lector humano.

3.12. Principios

Dublin Core conlleva tres principios, que se suponen críticos para entender cómo se debe pensar sobre las relaciones de los metadatos con los recursos subyacentes que describen:

- **Principio de correspondencia Uno a Uno:** En general, los metadatos Dublin Core describen una manifestación o una versión de un recurso, más que asumir que las manifestaciones sustituyen una a la otra. Por ejemplo, una imagen jpeg de la Mona Lisa tiene mucho en común con la pintura original, pero no es lo mismo que la pintura. De esta forma la imagen digital debería describirse tal y como es, probablemente con el creador de la imagen digital como Creador o Contribuidor, más que el pintor de la Mona Lisa original. La relación entre los metadatos para el original y para la reproducción es parte de la descripción de metadatos, y ayuda al usuario a determinar si necesita ir al Louvre para ver el original, o su necesidad puede satisfacerse con una reproducción.
- **El principio de simplificación o mutismo:** La cualificación de las propiedades del Dublin Core se rige por una regla, conocida como principio *Dumb-Down* (expresión del lenguaje coloquial para determinar que algo complejo se simplifica para hacerlo más fácil de entender). De acuerdo a esta regla, un cliente debería poder ignorar cualquier cualificador y utilizar el valor como si estuviera sin cualificar. Aunque esto puede conllevar algunas veces la pérdida de especificidad, el valor del elemento que permanece (sin el cualificador) puede seguir siendo correcto y útil para la localización/recuperación. La cualificación se considera, por tanto, sólo para matizar, no para extender el alcance semántico de una propiedad.
- **Valores apropiados:** La mejor práctica para un elemento particular o cualificador, puede variar por el contexto, pero normalmente un implementador no puede predecir siempre que el que va a interpretar los metadatos será siempre una máquina. Esto puede imponer ciertas restricciones en la forma de construir los metadatos, pero la necesidad de utilidad para la localización/recuperación debería tenerse en cuenta.

3.13. Modelo Abstracto Dublin Core

El propósito principal del modelo abstracto es proveer un marco de referencia contra el que se puedan comparar guías de codificación DC particulares. Para que funcione, un modelo de referencia debe ser independiente de cualquier sintaxis de codificación particular. Este modelo de referencia nos brinda un mejor entendimiento de los tipos de descripciones que tratamos de codificar y facilita el desarrollo de mejores correspondencias y traducciones entre diferentes sintaxis. A continuación, vamos a describir el modelo abstracto de los recursos y los conjuntos de descriciones, aunque el modelo abstracto puede consultarse de forma completa en el sitio web oficial de Dublin Core (http://dublincore.org).

El modelo abstracto de los recursos que se están describiendo presenta las propiedades que se enumeran a continuación:

- Cada *recurso* tiene cero o más pares *propiedad/valor*.
- Cada *par propiedad/valor* está compuesto por una propiedad y un valor.
- Cada *valor* es un *recurso* (la entidad física o conceptual que está asociada con la *propiedad* cuando se usa para describir un *recurso*).
- Cada *valor* puede ser un *literal* o no. Un valor no literal es un *valor* que es una entidad física, digital o conceptual.

- Un *literal* es una entidad que usa una cadena Unicode como forma léxica, en conjunción con una etiqueta opcional de lenguaje o tipo de datos, para denotar un recurso.

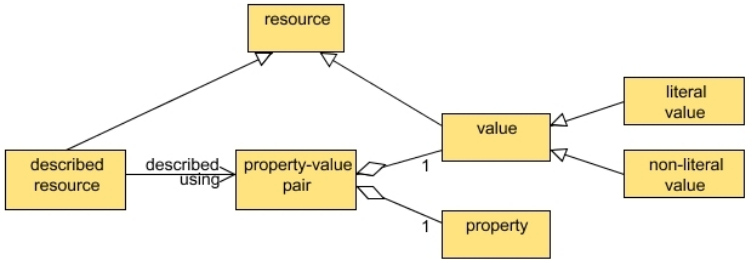


Figura 3.3: Modelo abstracto para los recursos

EL modelo abstracto de la DCMI para las descripciones de metadatos, reflejado en la figura 3.4, sigue las siguientes convenciones:

- Un *conjunto de descripciones* está compuesto por una o más descripciones, cada una de las cuales describe un único *recurso*.
- Una *descripción* está compuesta por una o más *sentencias* sobre uno y sólo un *recurso* y opcionalmente, la URI del *recurso* que se está describiendo.
- Cada *sentencia* instancia un par propiedad/valor, y está compuesta por una URI que identifica la propiedad y un valor
- Cada *valor* puede tener una URI del esquema de sintaxis de codificación.
- Cada *valor* puede tener asociado un valor de idioma dado por una etiqueta ISO.
- Cada *representación rica* es un texto marcado, una imagen, vídeo, audio, etc. o alguna combinación de las anteriores

3.14. Sintaxis de codificación Dublin Core

El modelo abstracto descrito anteriormente indica que cada descripción de metadatos DCMI describe uno y sólo un recurso. Sin embargo, las aplicaciones reales tienden a estar basada en conjuntos de descripciones estrechamente ligadas de alguna forma. Por ejemplo, un conjunto de descripciones puede comprender descripciones para una obra y el artista que la pintó. Además, a menudo se da que un conjunto de descripciones contenga alguna descripción acerca del conjunto en sí mismo (al cual nos podríamos referir como meta-metadatos).

Con la finalidad de poder intercambiar información entre aplicaciones software, los conjuntos de descripciones se instancian en la forma de registros de metadatos, de acuerdo a alguna guía de codificación DCMI. Cada guía de codificación propuesta da una serie de pautas sintácticas para representar cada conjunto de descripciones. Veamos a continuación las recomendaciones que propone

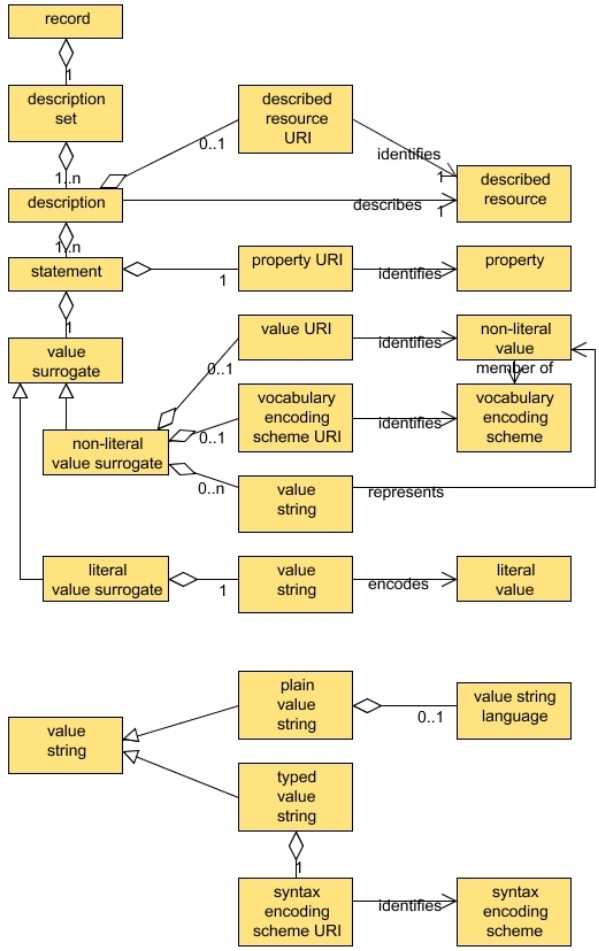


Figura 3.4: Modelo abstracto para los conjuntos de descripciones

DCMI (XML, RDF, XHTML) y por último otra codificación más reciente (Microformatos), y estrechamente ligada con la web semántica.

3.14.1. XML

En primer lugar veremos cómo codificar Dublin Core Simple en registros XML ”planos”. Para más información acerca de cómo embeber Dublin Core cualificado en XML puede visitar [Cor07].

De forma general, como recomendación de DCMI la información DC plasmada como XML debería basarse en Schemas XML más que en DTD’s, ya que los enfoques basados en Schemas XML suelen ser más flexibles y más reutilizables. Además, es deseable usar espacio de nombres (*namespaces*) para identificar de forma inequivoca los elementos DC, refinamientos de elementos y esquemas de codificación.

Por otra parte, hay que hacer notar que, como documento XML bien formado, debería tener una raíz que actúe de contenedor de los elementos codificados. La recomendación no hace referencia al nombre de este elemento contenedor ni el espacio de nombres del que se podría sacar. Sin embargo, como posibles candidatos podríamos tener las etiquetas: *dc, dublinCore, record o metadata*.

Como describe el modelo abstracto de Dublin Core simple, podemos tener las siguientes entidades:

- Un registro DC simple está compuesto por una o más propiedades y sus valores asociados.
- Cada propiedad es un atributo del recurso que se está describiendo.
- Cada propiedad debe se una de las dadas por el conjunto de elementos de metadatos Dublin Core.
- Las propiedades se pueden repetir.
- Cada valor es un literal (*string*).
- Cada valor puede tener un idioma asociado.

Formalmente hablando, no hay conexión entre el registro DC simple y el recurso que se está describiendo. Para ello, la conexión se podría dar (aunque no obligatorio) codificando la URI del recurso como valor del elemento DC. *Identifier*.

Siguiendo con las recomendaciones, como dice DCMI, las propiedades se deberían codificar como elementos XML (en minúsculas) y los valores como el contenido de estos elementos. El nombre del elemento XML debe ser un nombre XML cualificado que asocia un nombre de elemento con el apropiado del espacio de nombres DCMI, como muestra el siguiente ejemplo:

```
<dc:title>Dublin Core en XML</dc:title>
```

en vez de:

```
<dc:title value="Dublin Core en XML" />
```

Para codificar múltiples valores para una propiedad dada, simplemente hay que repetir el elemento XMl para esa propiedad:

```
<dc:title>Mi primer titulo</dc:title>
<dc:title>Mi segundo titulo</dc:title>
```

En caso de que se vaya a especificar el idioma usado en el valor para una propiedad, se debe usar el atributo *xml:lang*:

```
<dc:subject xml:lang="es">marisco</dc:subject>
<dc:subject xml:lang="fr">fruits de mer</dc:subject>
```

3.14.2. RDF

La plataforma para la descripción de recursos (RDF - *Resource Description Framework*) es hoy en día uno de los pilares principales, junto con el lenguaje XML, de la Web Semántica. RDF tiene en sus orígenes a Cyc: un proyecto para crear crear inteligencia artificial con un ”sentido común” básico. Consta de una gran base de datos que contiene definiciones tales como por ejemplo ”un árbol es una especie de planta, un sauce es una especie de árbol, etc”. Cyc puede por lo tanto deducir de estas definiciones que un sauce es una planta.

RDF es un lenguaje general de representación de información. Por decirlo de otro modo es lo que podríamos denominar un lenguaje de metadatos. Es un estándar del W3C por lo que tiene un potencial inmenso, aunque es más común encontrar ejemplos de RDF Site Summary que de RDF propiamente dicho y además, en general un fichero RDF no se muestra a personas.

RDF se puede ver como la respuesta de la informática a la necesidad de disponer de sistemas altamente homogéneos para describir recursos digitales en particular y recursos de cualquier tipo en general. RDF no es un conjunto de metadatos predefinidos, como Dublin Core, sino un conjunto de especificaciones para crear cualquier tipo conjunto de metadatos siguiendo un conjunto de normas pre-establecidas que hacen posible el intercambio de descripciones creadas por agentes distintos y sin necesidad de que se hayan puesto previamente de acuerdo entre ellos.

Para embeber los elementos de Dublin Core en XML usando RDF, se debe proporcionar una DTD o un XML Schema con el cual poder validar los documentos generados. Cualquier documento XML bien formado debe incluir una sentencia acerca de la versión usada de XML. Actualmente, la única versión válida de XML (dada en la recomendación W3C) es 1.0. Por tanto, el documento deberá empezar por:

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

Seguidamente la DTD usada se referencia mediante:

```
<!DOCTYPE rdf:RDF PUBLIC "-//DUBLIN CORE//DCMES DTD 2002/07/31//EN"
    "http://dublincore.org/documents/2002/07/31/dcmes-xml/dcmes-xml-dtd.dtd">
```

Es necesario declarar que se está usando RDF, para que las aplicaciones puedan reconocerlo como un documento RDF/XML. La siguiente sentencia declararía la etiqueta de apertura con su espacio de nombres XML y el espacio de nombres usado para los elementos DC:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
```

La codificación permite describir múltiples recursos en un único documento. Cada recurso descrito viene encerrado en un elemento contenedor etiquetado como *rdf:Description*. Dentro de esta etiqueta, se coloca cada elemento precedido por el espacio de nombres *dc:*. Por ejemplo, el elemento *Title* produce la etiqueta (completamente en minúsculas) *dc:title* dentro del contenedor de la descripción como sigue:

```
<rdf:Description rdf:about="http://www.virtualmuseum.com/">
  <dc:title>Virtual Museum</dc:title>
</rdf:Description>
```

Si el valor de algún elemento DC es un recurso que tiene una URI en vez de texto plano, se debería describir el valor del atributo *rdf:resource* en la etiqueta, con un contenido vacío:

```
<rdf:Description rdf:about="http://www.virtualmuseum.com/">
  <dc:source rdf:resource="http://www.virtualmuseum.com/otradireccion/"/>
</rdf:Description>
```

De esta forma, se puede repetir esta estructura para otros elementos del conjunto DC. Hay que hacer notar que el conjunto de elementos dentro de la descripción, como conjunto, no presenta orden alguno y que por tanto las aplicaciones que consuman el documento no tienen porqué preservar el orden de los elementos dentro del contenedor de la descripción.

Para describir el idioma usado para el contenido del elemento, XML provee el atributo *xml:lang* en el cual se puede embeber dicho idioma.

```
 <rdf:Description rdf:about="http://dublincore.org/">
   <dc:title xml:lang="fr">L'Initiative de métadonnées du Dublin Core</dc:title>
   <dc:title xml:lang="de">der Dublin-Core Metadata-Diskussionen</dc:title>
 </rdf:Description>
```

```
<head>
  <link rel="meta" href="nombre_pagina.rdf" />
</head>
```

3.14.3. HTML/XHTML

Otra forma de especificar registros de metadatos es embeberlos en páginas HTML/XHTML. Como hemos visto anteriormente, se puede crear un archivo RDF o XML separado y enlazarlo usando etiquetas HTML/XHTML <*link*>. En este apartado veremos cómo embeber información DC a través de etiquetas <*meta*> y <*link*>. Como indica la recomendación, sólo es posible describir un recurso único usando este método. Para describir múltiples recursos se puede cualquiera de los dos métodos anteriores.

Entre sus ventajas se tiene que la utilización de etiquetas <*meta*> aparecieron en las primeras especificaciones de HTML, son ampliamente conocidas y tienen un uso extendido. Sin embargo, debido a su extensión es frecuente ver su excesivo uso con propósitos no correctos, y además, no son interpretables directamente por un usuario.

Los metadatos DC a través de este método se embeben dentro de la sección <*head*> de una página XHTML. Para codificar un elemento DC, se usa el atributo *name* y *content* dentro de las etiquetas XHTML *meta*:

```
<meta name="DC.element" content="Value" />
<meta name="DCTERMS.element" content="Value" />
```

De forma general, los nombres de elementos pueden contener tanto mayúsculas como minúsculas pero deberían presentar siempre la primera letra como minúscula. El valor del atributo *content* se define como CDATA; es decir, una secuencia de caracteres dado por el conjunto de caracteres dado por el documento.

Para indicar el esquema de codificación se usa el atributo *scheme* dentro de la etiqueta <*meta*>. El esquema de codificación usado debería seguir los nombres especificados por la recomendación de términos DCMI, y pueden presentar mayúsculas y minúsculas empezando por mayúscula, aunque a menudo se especifican escribiendo la palabra en mayúsculas completamente.

```
<meta name="DC.date" scheme="DCTERMS.W3CDTF" content="2006-10-26" />
<meta name="DC.type" scheme="DCTERMS.DCMIType" content="Text" />
```

Cuando el valor de una propiedad es la URI de otro recurso (como se podría dar en el caso de un elemento DC.relation), se prefiere una forma alternativa de codificación dada la etiqueta XHTML *<link>*:

```
<link rel="propiedad" href="URIdelRecurso" />
```

En los documentos donde se indique el idioma usada para completar un elemento, se debería usar el atributo *xml:lang* de la etiqueta XHTML *<meta>* y/o el atributo *hreflang* de la etiqueta *link*. Por ejemplo:

```
<meta name="DC.subject" xml:lang="en-GB" content="seafood" />
<meta name="DC.subject" xml:lang="fr" content="fruits de mer" />
<link rel="DC.relation" hreflang="en" href="http://www.virtualmuseum.com/en/" />
<link rel="DC.relation" hreflang="de" href="http://www.virtualmuseum.com/de/" />
```

Los prefijos *DC* y *DCTERMS* usados en los nombres de las propiedades durante este apartado se usan para indicar el espacio de nombres del que se considera la propiedad. La URI del *namespace* se debería codificar mediante la etiqueta *<link>*, usando el siguiente patrón:

```
<link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
<link rel="schema.DCTERMS" href="http://purl.org/dc/terms/" />
```

3.14.4. Microformato Dublin Core

Los microformatos aprovechan características de HTML/XHTML para añadir información semántica, e intentan ser útiles principalmente a las personas, y en segundo lugar a los agentes de software (por ejemplo, los buscadores y otros más avanzados). Aunque esta sintaxis está destinada principalemente a las personas, en contra tiene que son menos potentes que las otras ya que por ejemplo no permiten definir formalmente relaciones complejas que puedan servir para que los agentes de software realicen inferencias o deducciones.

En general, los microformatos consisten en soluciones estándar de marcado HTML/XHTML que utilizan las características semánticas que ofrecen las propiedades *class* de cada elemento para casos de uso concretos. Hay estructuras de marcado HTML que son rápidamente aceptadas como ”buena práctica” y recomendadas y adoptadas de forma general, tales como lista desordenadas para construir menús o listas de definición para fichas de elementos de algún tipo. Una vez que se abstraen estos patrones comunes, se pueden formalizar e identificar, por ejemplo, por medio del atributo *class*. Particularizando para el caso de Dublin Core, el esquema de codificación mediante microformatos vendría dada a través de lista de definición, como muestra la siguiente porción de código HTML:

```
<dl class="dublincore">
<dt>Título:</dt>
<dt>Título</dt>
<dd class="title">El ingenioso hidalgo Don Quijote de la Mancha</dd>

<dt>Autor:</dt>
<dd class="creator">Miguel de Cervantes Saavedra</dd>

<dt>Fecha de creación:</dt>
<dd class="date">1604</dd>
</dl>
```

En este caso, los metadatos Dublin Core se embeben en un a lista de definición *dl*. Este elemento tiene una clase *dublincore*, que podría tener asociado diversas propiedades para su representación por medio de hojas de estilo CSS, por ejemplo. Sin embargo, donde radica la idea de los microformatos es que además de forma, puede tener contenido semántico. Dentro de la etiqueta, se encuentra la clase *title* que es uno de los elementos Dublin Core. De forma similar, se podría embeber la información oportuna dentro del resto de elementos de la lista de definición.

La principal ventaja que presenta esta sintaxis de codificación respecto a las anteriores, es que son visibles directamente por personas. De hecho, con los microformatos se pretende facilitar el acceso a la información contenida en los metadatos para las personas, por encima de los agentes de usuario. Como desventaja de este esquema de codificación nos encontramos con que su uso todavía no ha sido adoptado ni por desarrolladores de aplicaciones, ni por buscadores y de momento, está lejos de convertirse en un estándar de facto. Si los microformatos de Dublin Core fuesen un estándar el código anterior podría ser recogido por los navegadores e indexado siguiendo el conjunto de elementos Dublin Core para su posterior consulta.

PARTE II

DESARROLLO DEL SISTEMA

Capítulo 4

PLAN DE DESARROLLO DEL PROYECTO

No basta tener buen ingenio; lo principal es aplicarlo bien.

René Descartes

Índice del Capítulo

4.1. Introducción

Este documento describe el plan de desarrollo general que se seguirá para el desarrollo del proyecto Sistema de generación de Museos Virtuales. El motivo de realizar este plan de desarrollo se debe a que la calidad en el ámbito del desarrollo software no sólo se debe enfocar en el producto terminado, sino también en el proceso que se sigue hasta llegar al resultado final.

Este plan del proyecto tiene entre sus objetivos definir las actividades necesarias para la construcción del proyecto Sistema de generación de Museos Virtuales. Los planes trazados en este documento están basados en los requisitos del proyecto.

4.2. Perspectiva general del proyecto

4.2.1. Metodología

Inicialmente la metodología usada ha seguido las pautas del Proceso Unificado. El Proceso Unificado es un proceso de desarrollo software iterativo e incremental, dirigido por los casos de uso y centrado en la arquitectura, que plantea un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. Además de ello, el Proceso Unificado puede verse como un marco de trabajo que debe ser adaptado en una serie de variables: tamaño del sistema, dominio en el que va a trabajar, complejidad y nivel de proceso de la organización del proyecto.

Adicionalmente a las guías demarcadas por el Proceso Unificado y dadas las circunstancias en las que se iba desarrollando el proyecto, se optó por introducir los principios propuestos por el Modelado Ágil (*Agile Modeling*) de Scott Ambler. El punto clave en el Modelado Ágil se basa en sus prácticas y principios culturales, que animan a los desarrolladores a producir suficientes modelos de forma que soporten los problemas de diseño y documentación de forma adecuada. Un concepto importante del Modelo Ágil es que no se trata de un proceso de software completo, por lo que se necesitará concretar el proceso con otras metodologías como XP, DSDM, SCRUM o en nuestro caso: el Proceso Unificado.

Sumando estos dos enfoques se podría decir que la metodología seguida para el desarrollo del proyecto se basa en el Proceso Unificado como marco de trabajo, y adaptado minimizando los artefactos a desarrollar y la cantidad de documentos a mantener, según los principios de mantener la documentación simple, aceptar los cambios, modelar con un propósito, y agilizar los flujos de trabajo manteniendo un viaje ligero a lo largo del desarrollo del producto.

4.2.2. Referencias

Principalmente, las fuentes de información adicionales a este documento vienen dadas por la metodología usada, por lo que se propone seguir las referencias bibliográficas [JR00b] y [Amb02], así como las normas IEEE para la especificación de requisitos software 830 y planificación de proyectos 1058.1.

4.2.3. Elementos a entregar del proyecto

A continuación se indican y describen cada uno de los artefactos que serán generados a través de las diferentes fases e iteraciones y utilizados por el proyecto y que constituyen los entregables. Es preciso destacar que debido a la naturaleza iterativa e incremental del proceso, todos los artefactos son objeto de modificaciones a lo largo del desarrollo, con lo cual los presentados en esta memoria serán la versión definitiva y completa de cada uno de ellos:

- **Requisitos**: aúna al documento de especificación y descripción de requisitos software, así como el modelo de casos de uso.
- **Análisis**: abarca al diagrama de clases de análisis, y realizaciones de casos de uso en colaboraciones.
- **Diseño**: comprende al diagrama de clases de diseño, realizaciones de casos de uso, diagramas de actividad de operaciones, diseño relacional del esquema de la base de datos, diccionario de datos.
- **Implementación**: la jerarquía de ficheros de código fuente cuya compilación producen el sistema ejecutable.
- **Despliegue**: el conjunto de componentes software junto con la documentación necesaria para desplegarlo en producción, así como el conjunto de rutinas para instalar, inicializar y usar el sistema en explotación.
- **Plan de iteraciones**: planificación temporal y de actividades, así como los hitos a alcanzar en cada una de las iteraciones.
- **Documento de métricas**: informes de diversas métricas obtenidas a partir del código fuente. Principalmente se elaborarán las métricas mediante PMD (*http://pmd.sourceforge.net*), CPD (*http://pmd.sourceforge.net/cpd.html*), y FindBugs (*http://findbugs.sourceforge.net*)

4.3. Organización del proyecto

4.3.1. Estructura organizativa

Esta sección describe la estructura organizativa de roles desempeñados, junto con sus actividades asociadas, a lo largo del desarrollo del proyecto, y se muestra gráficamente en la figura 4.1:

1. **Jefe de Proyecto**
   - Planificar el proyecto.
   - Supervisar el desarrollo del mismo y gestionar los tiempos de cada iteración, evaluando los hitos alcanzados.
2. **Ingeniero de Casos de Uso**

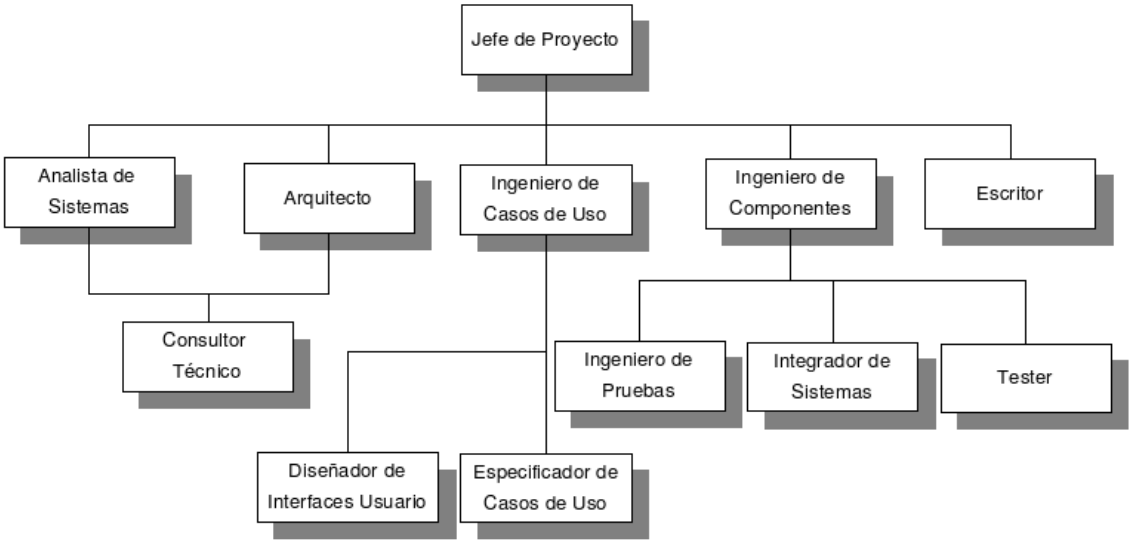


Figura 4.1: Estructura organizativa por roles

- Análisis y diseño de los casos de uso y sus realizaciones.
- Asegurar que la realización de cada caso de uso lleva a cabo correctamente el comportamiento de su correspondiente caso de uso, y sólo ese comportamiento.

3. **Analista de Sistemas**

   - Dirigir el modelado.
   - Delimitar el alcance y límites del sistema a desarrollar, encontrando actores y casos de uso, para producir un modelo de casos de uso y modelo de dominio.
   - Elaborar el conjunto de requisitos funcionales y no funcionales.
   - Identificar las clases de análisis y sus relaciones.

4. **Arquitecto**

   - Dirigir el diseño técnico.
   - Describir la vista de arquitectura del modelo de casos de uso, análisis, diseño y despliegue.
   - Asistir al integrador de sistemas en la elaboración de los procedimientos para la construcción, instalación y despliegue del sistema.
   - Elaboración del modelo de diseño.

5. **Ingeniero de Componentes**

   - Asegurar que cada clase de análisis cumple los requisitos esperados en el modelo de análisis.

- Completar las operaciones y atributos de las clases de diseño propuestas por el arquitecto.
- Garantizar que cada clase de diseño cumple los requisitos esperados según las realizaciones de casos de uso.
- Implementar los artefactos del modelo de diseño.

6. **Diseñador de Interfaces de Usuario**
   - Dar forma visual a las interfaces usuario en cuanto a su maquetación, interacción y experiencia.
7. **Especificador de Casos de Uso**
   - Asistir al analista de sistemas en la especificación detallada de los casos de uso.
8. **Consultor Técnico**
   - Resolver las dudas técnicas de los ingenieros de componentes y asistir al arquitecto y analista sobre cuestiones relacionadas con el uso de nuevas tecnologías.
   - Plantear *spikes* a modo de pruebas de concepto plasmando el correcto de uso de funciones reducidas que serán necesarias dentro del sistema completo.
9. **Ingeniero de Pruebas**
   - Planear las pruebas y sus objetivos.
   - Elaborar los procedimientos de prueba necesarios para su realización por parte del tester.
10. **Tester**
    - Realizar las pruebas planeadas para cada entrega del sistema.
    - Asegurar el funcionamiento del sistema a partir de los manuales de usuario, y aportar consideraciones acerca de la experiencia del uso.
11. **Integrador de Sistemas**
    - Planificar la secuencia de operaciones necesarias para la construcción, instalación y despliegue del sistema tanto en el ambiente de desarrollo como el de explotación.
12. **Escritor**
    - Asiste al resto de roles en la elaboración de las memorias que registran los requisitos, documentación técnica, reuniones, hitos y decisiones alcanzados durante el desarrollo del sistema.
    - Produce los manuales de usuario.

4.3.2. Interfaces externas

El responsable de desarrollo se comunicará con los responsables y tutores del proyecto, Pablo de la Fuente y Jesús Vegas, ambos Doctores en Informática por la Universidad de Valladolid, para la definición de requisitos, revisión y tutela de los hitos alcanzados al final de cada iteración, así como para asegurar la calidad del desarrollo.

4.3.3. Responsabibilidades

Debido a que el presente sistema se desarrolla como proyecto fin de carrera, todos los roles de la estructura organizativa recaen sobre el mismo autor del proyecto.

4.4. Gestión del proyecto

4.4.1. Gestión de riesgos

El modo en que se planifica el desarrollo del sistema está condicionado en gran parte por los riesgos que se perciben. Por tanto, se comenzará con una identificación de los posibles riesgos que pueden ocurrir a lo largo del desarrollo del sistema, evaluando su impacto y a continuación se planteará un plan de contingencia que indica lo que ha de hacerse en caso de que el riesgo se materialice:

1. **No conseguir la arquitectura correcta**
   No construir un sistema que pueda evolucionar suavemente por las fases siguientes o durante su tiempo de vida compone uno de los riesgos más serios, que suelen ocurrir durante las fases de inicio y elaboración.

   Plan de contingencia: Identificar los casos de uso críticos y requisitos no funcionales importantes, para encontrar el esqueleto sobre el que apoyar el sistema completo. Además, se seguirá la práctica de construir y liberar más a menudo para evaluar si el producto que se va desarrollando cumple con los requisitos.

2. **No conseguir los requisitos correctos**
   No conseguir que el sistem haga lo que los usuarios esperan realmente de él por una captura de requisitos deficiente.

   Plan de contingencia: Asegurar al final de cada iteración si el comportamiento diseñado e implementado de cada caso de uso cumple con su especificación inicial.

3. **Modificar y ampliar los requisitos**
   Relacionado con el riesgo anterior, los cambios en los requisitos pueden llevar a no desarrollar el sistema que los usuarios estaban esperando.

   Plan de contingencia: En cualquier caso, si el sistema desarrollado se ajusta a los requisitos iniciales, éste constituirá un producto en sí mismo que hace honor a dicho conjunto de especificaciones. Cualquier cambio y ampliación venidera sobre el conjunto de requisitos inicial será tratado (evaluando su impacto y dependencias) ya que en la mayoría de los casos, esto presentará un valor añadido sobre el producto desarrollado. Por eso, se aplicará la práctica ágil de permitir y aceptar los cambios como un suceso natural y frecuente en el desarrollo de sistemas software.

4. **Uso de nuevas tecnologías**
   El ambiente técnico en el que se desarrolla este proyecto se compone de tecnologías relativamente desconocidas por lo que la estimación de tiempos puede ser poco realista.

Plan de contingencia: Plantear el rol de Consultor Técnico desde el inicio del proyecto para contemplar las actividades de realización de *spikes* que permitan orientar en la planificación el coste que conlleva la implementación de ciertas funcionalidades.

5. **Variación de los recursos disponibles**
   Existe cierta probabilidad de que la dedicación de tiempo al proyecto por parte de los recursos cambie por su implicación en otros proyectos ajenos al presente, por lo que en este caso la planificación no puede mantenerse.

   Plan de contingencia: Como los recursos para el proyecto son limitados, no se pueden incorporar más recursos para mitigar su impacto, por lo que la aparición de este riesgo conlleva alargar el tiempo planificado para cada una de las tareas no completadas desde el momento en que surgió.

6. **Desvios en la planificación**
   Relacionado con el riesgo del uso de nuevas tecnologías, este riesgo se produce cuando se da una planificación no realista del proyecto, debido a que se han subestimado las tareas. La planificación de tareas que se da al inicio del proyecto son estimaciones que se dan a futuro, por lo que los desvíos que se produzcan en este sentido pueden aparecer con frecuencia.

   Plan de contingencia: Evaluar la situación del proyecto frecuentemente. En primer lugar, se evaluará si se van cumpliendo los hitos marcados para las iteraciones, y en segundo lugar habrá que revisar el trabajo personal realizado con más frecuencia que la dada por el plan de iteraciones para averiguar si la lista de subtareas se va cumpliendo y qué problemas se han encontrado.

7. **Incidencias hardware y software**
   Las averías tanto software como hardware de las herramientas y equipos utilizados puede conllevar serias pérdidas de información y avances del proyecto.

   Plan de contingencia: Ante la posible avería hardware de un equipo, la solución consiste en reemplazar el componente con defectos o el equipo entero. Ante una pérdida de datos, hemos de anticiparnos proponiendo un sistema para asegurar la redundancia de datos. Para conseguirlo, se puede plantear un esquema de copias de seguridad periódicas y voluntarias de los directorios de trabajo del proyecto, y la introducción de un sistema gestor de versiones para mantener además la historia de cada archivo de código fuente, así como permitir una rápida gestión y recuperación a partir de las versiones almacenadas en el repositorio. Además, puede ser adecuado mantener dichas copias de datos en otro dispositivo independiente físicamente.

4.4.2. Mecanismos de supervisión y control

Desde el punto de vista de seguimiento del proyecto, se mantendrán diversas reuniones con los tutores del proyecto. En estas reuniones se realizará el control de las actividades realizadas hasta el momento, evaluación del producto actual, y actividades a realizar a partir de la reunión, revisando, si es necesario, la planificación original.

Además de ello, para mantener el control sobre los cambios y mantener la integridad entre los diversos artefactos se tendrán en cuenta dos ideas. Por una parte, se intentará seguir la práctica dada por el modelado ágil de minimizar la documentación a mantener haciendo que cada item de información

del proyecto resida en un único punto de forma que mantenimiento sea más sencillo. Y por otra parte, como conseguir un acoplamiento mínimo entre modelos, código y documentación en general es complicado, se harán revisiones frecuentes para realimentar y evolucionar cada artefacto entregable del proyecto.

4.4.3. Estimaciones del proyecto

Las circunstancias bajo las cuales se desarrolla este proyecto obliga a que su coste sea nulo, por lo que no deberán producir estimaciones ni reestimaciones en este sentido.

4.4.4. Plan del personal

El proyecto a desarrollar cuenta con un número de recursos humanos reducido (dado que representa el trabajo de un proyecto fin de carrera), y que se limita al propio autor del mismo. Por tanto, todas las tareas planificadas se llevarán a cabo por la misma persona.

Desde la perspectiva de seguimiento de la la planificación que se realizó inicialmente, reflejada en el apartado 4.6, conviene señalar que ésta se vió vista obligada a variar a lo largo de las iteraciones. Si bien, ya se contaba con riesgos que pudieran hacer que la estimación no fuera del todo precisa, hay que que hacer notar que las condiciones laborales del autor cambiaron prácticamente al inicio del proyecto, por lo que su dedicación se vió forzada a disminuir drásticamente, lo que produjo una ampliación muy considerable del tiempo de entrega del mismo.

4.5. Proceso técnico

4.5.1. Herramientas

El proceso de desarrollo del producto se desarrollará de forma casi completa bajo una máquina GNU/Linux (kernel 2.6.12). Desde el punto de vista de herramientas de documentación, se usarán las siguientes aplicaciones a lo largo de las diferentes fases del proyecto:

- Análisis: Together Designer 2005 (*bajo Windows XP*)
- Diseño: Together Architect 2006 (*bajo Windows XP*)
- Diagramas no UML de apoyo: Open Office 1.1, y Edraw Max v4 (*bajo Windows XP*)
- Memoria: LaTeX3.14 mediante el editor Kile 1.8

Para la implementación del sistema, se hará uso de las siguientes herramientas:

- Contenedor web Apache Tomcat 6.0.16

- Servidor de base de datos MySQL 4.0.24
- Sun JDK 1.6.0
- Kile 1.8
- Eclipse 3.1
- Bluefish 1.0
- Vim-Gtk 6.3
- Ant 1.6.5
- Subversion 1.2

4.5.2. Documentación del software

Se recomienda encarecidamente incluir dentro del propio software desarrollado comentarios como fuente de información que den un valor añadido a la documentación general. La recomendación se da por varias razones: primero, porque el código escrito por un programador lo leerán posiblemente otros programadores, o bien él mismo tiempo después, y además, porque los comentarios de las tareas concretas de una rutina o algoritmo en el propio fuente hace que su mantenimiento se dé en este punto: si el algoritmo (o parte de él) desaparece sus comentarios lo harán con él, y no habrá que mantener por una parte el código y por otra la documentación del mismo. A pesar de ello, no hay que perder de vista que estos comentarios consistirán en apuntes de bajo nivel de la funcionalidad concreta, y que en ningún caso constituirá información de alto nivel.

Asimismo, también hay que hacer notar que será particularmente apropiado el uso de herramientas de documentación estándar y de uso extendido dentro del marco de desarrollo del proyecto y su comunidad. De esta forma, al tratarse de un sistema construido bajo la plataforma Java EE, se recomienda el uso de la herramienta JavaDoc.

4.6. Planificación

Fase de Inicio			
Iteración	**Descripción**	**Hito asociado**	**Estimación**
1	▪ Elaborar el plan de proyecto ▪ Marco general del proyecto	Plan de proyecto inicial	2 semanas

Cuadro 4.1: Tareas de la fase de inicio

Fase de Elaboración			
Iteración	**Descripción**	**Hito asociado**	**Estimación**
1	▪ Definir y elaborar los casos de uso ▪ Desarrollar el modelo de análisis (clases principales del dominio) ▪ Plantear bocetos de pantallas como medio de captura de requisitos preliminar	▪ Análisis conceptual preliminar ▪ Bocetos de pantallas principales	2 semanas
2	▪ Construir *spikes* con aspectos funcionales que aclaren el uso de nuevas tecnologías ▪ Colaboraciones de clases de análisis ▪ Especificaciones suplementarias ▪ Planteamiento de la arquitectura	▪ Modelo de análisis ▪ Modelo de diseño preliminar	2 semanas
3	▪ Elaborar el modelo de diseño ▪ Especificaciones suplementarias ▪ Diseño de la arquitectura ▪ Realizaciones de casos de uso como diagramas de secuencia y colaboración	▪ Prototipo de arquitectura ▪ Modelo de diseño ▪ Diagramas de interacción para la realización de casos de uso en diseño	2 semanas

Cuadro 4.2: Tareas de la fase de elaboración

Fase de Construcción			
Iteración	**Descripción**	**Hito asociado**	**Estimación**
1	▪ Elaborar navegación entre pantallas aplicación administrador	Pantallas enlazadas de aplicación administrador	1 semana
2	▪ Modelado de objetos y relacional de las entidades del sistema ▪ *Mapeos* Hibernate ▪ Autenticación y autorización de accciones ▪ Pruebas y correcciones	▪ Capa de acceso a datos inicial	3 semanas
3	▪ Implementación ▪ Pruebas y correcciones ▪ Diagramas de actividad	▪ Aplicación administrador navegable	2 semanas
4	▪ Implementación ▪ Pruebas y correcciones	▪ Aplicación responsable del museo navegable ▪ Capa de lógica de negocio para administrador	3 semanas
5	▪ Implementación ▪ Pruebas y correcciones	▪ Capa de acceso a datos ▪ Capa de lógica de negocio para responsable del museo	5 semanas
6	▪ Implementación ▪ Pruebas y correcciones	▪ Vista web para visitante	2 semanas
7	▪ Construcción de *spikes* para explorar las dificultades del uso de servicios de terceros ▪ Implementación ▪ Pruebas y correcciones	▪ Servicios Web 2.0	3 semanas
8	▪ Implementación ▪ Pruebas y correcciones	▪ Capa de servicios ▪ Vista móvil para visitante	2 semanas

Cuadro 4.3: Tareas de la fase de construcción

Fase de Transición			
Iteración	**Descripción**	**Hito asociado**	**Estimación**
1	▪ Puesta en marcha del producto en ambiente de desarrollo ▪ Elaboración de manual de instalación ▪ Elaboración de manuales de usuarios	Producto final en ambiente de desarrollo	2 semanas
2	▪ Implantar producto en ambiente de explotación ▪ Asegurar funcionamiento global	Producto final en ambiente de explotación	1 semana

Cuadro 4.4: Tareas de la fase de transición

Capítulo 5

ESPECIFICACIÓN DE REQUISITOS

Si tu intención es describir la verdad, hazlo con sencillez
y la elegancia déjasela al sastre.

Albert Einstein

Índice del Capítulo

5.1. Introducción

5.1.1. Propósito

El propósito de este capítulo consiste en especificar los requisitos de un sistema gestor de museos virtuales. El público objetivo del mismo incluye principalmente al analista de sistemas, arquitecto y al cliente.

5.1.2. Alcance

La motivación principal para el desarrollo de este sistema es proporcionar una herramienta a los responsables de pequeños y medianos museos que les permita la publicación *on-line* de información relativa de los objetos que exponen.

Esta herramienta permitirá que los responsables de museos puedan gestionar los contenidos que deseen publicar de una manera sencilla y que no les obligue a mantener un servidor dedicado.

5.1.3. Referencias

La principal referencia de este documento se basa en la norma IEEE 830 para la especificación de requisitos software.

5.1.4. Perspectiva general

El resto del capítulo se divide en dos partes principalmente: una primera sección que contiene una descripción general y de alto nivel del sistema a desarrollar, y una segunda sección que comprende los requisitos específicos divididos por rol.

5.2. Descripción general

5.2.1. Perspectiva del producto

Las nuevas tecnologías introducen un nuevo mecanismo de publicación de contenidos a través de Internet. Esto favorece que los pequeños y medianos museos puedan dar a conocer a una comunidad de usuarios mucho más grande información acerca de los objetos que exponen.

Un sistema gestor de museos virtuales, como el que se va a construir, permite la creación, modificación, *hosting*, administración, comunicación y visualización de museos virtuales. Aunque puede enmarcarse dentro de los sistemas gestores de contenidos (CMS) genéricos, está orientado a la gestión de sitios web para pequeños y medianos museos de arte.

Si bien el usuario puede crear un sitio web genérico, encontrará más facilidades y funcionalidades para publicar sus obras, u objetos expuestos en general. Asimismo, siguiendo esta orientación hacia la gestión de colecciones de obras artísticas, el proyecto permitirá seguir el estándar de metadatos Dublin Core.

Aunque el proyecto pretende producir un sistema gestor de contenidos especializado, la visualización de los museos virtuales generados no se limita a su forma web clásica, sino que se dispondrá de una visualización para dispositivos móviles de ciertos contenidos elegidos por el usuario.

Además de esa vía alternativa de visualización, el museo virtual generado puede permitir exponer o acceder a ciertos servicios con los que interoperar tanto desde el punto de vista de lo que expone el museo virtual a terceros como las aplicaciones con las que puede comunicarse.

El sistema desarrollado producirá un producto autónomo: no se integrará ni dependerá de otros sistemas, a excepción de los componentes de ejecución básicos requeridos por la tecnología empleada (sistema operativo, máquina virtual java, contenedor servlet, etc).

El sistema contempla tres tipos de roles distintos: administrador, responsable del museo y visitante. Para cada rol el sistema debe aportar las interfaces adecuadas pudiéndose ver como un sistema dividido en cuatro aplicaciones diferentes que acceden a la misma información con unos privilegios determinados por el tipo de rol:

- **Administración (rol administrador):** Tiene los privilegios máximos del sistema pudiendo supervisar o modificar cualquier acción ejecutada por el resto de usuarios. La aplicación en sí misma dará acceso a la gestión de las entidades *maestras* del sistema completo.
- **Gestión de Museos Virtuales (rol responsable del museo o director):** Compone el núcleo del sistema permitiendo crear, modificar y mantener el museo virtual.
- **Sitio web del museo virtual generado (rol visitante):** Como resultado principal del trabajo del responsable del museo, se podrá publicar un sitio web para el visitante o público en general en Internet.
- **Vista para dispositivos móviles (rol visitante):** Además de la vista web, la oferta de contenidos para el visitante permitirá la visualización de ciertas secciones del museo virtual como una aplicación independiente al resto en un dispositivo móvil.

Interfaces de sistema

El sistema que se está realizando sigue una arquitectura extensible tipo *Cliente-Servidor*.

Interfaces de usuario

Todas las aplicaciones que producirá el sistema deberán estar disponibles a través de un navegador de uso común, por lo que se deberá fomentar la compatibilidad y seguimiento de los estándares comúnmente aceptados como buenas prácticas. La aplicación que genera una vista para móviles deberá poder ejecutable en un dispositivo móvil con soporte Java, y especificamente, compatible con MIDP.

Interfaces de hardware

Las aplicaciones de las que consta el sistema deben poder ser visualizadas al menos en un ordenador personal, a excepción de la vista móvil, que está destinada a ejecutarse en un dispostivo móvil.

Interfaces de software

El proyecto debe implementarse bajo la plataforma Java EE y la elección de componentes para armar la arquitectura debe basarse, entre otros factores, en su disponibilidad como software de libre distribución.

Operaciones

Las operaciones que se pueden realizar con este sistema deberán ser intuitivas para los usuarios. Por ello, este sistema debe estar creado buscando la facilidad de uso, de forma que no se necesite una formación adicional exhaustiva para su manejo o mantenimiento, teniendo en cuenta que sería deseable que el encargado del mantenimiento del sistema fuera una persona con conocimientos en redes y administración de sistemas Linux, tanto a nivel de servidores web como de base de datos.

Se deberán tener en cuenta además los requisitos de interacción comentados en el apartado 2.3.

5.2.2. Funciones del producto

En este apartado delimitaremos a grandes rasgos las funciones del sistema software a construir:

- Administración de usuarios
- Administración de museos virtuales
- Obtención de estadísticas
- Gestión de contenidos *on-line* del museo virtual
- Comunicación entre los tres roles del sistema
- Publicación y visualización de museos virtuales via web
- Publicación y visualización de museos virtuales en móviles o PDA's
- *Hosting* de museos virtuales
- Soporte para la inclusión de metadatos Dublin Core en los contenidos
- Intercambio de información entre el museo virtual y otros sistemas externos o servicios (Google Maps, Flickr, Picasa, RSS, XML y servicios web)

- Personalización de la apariencia de los museos virtuales generados
- Soporte de idiomas

5.2.3. Descripción del modelo de negocio

Aunque se describirá con mayor detalle, la figura 5.1 presenta una aproximación al modelo de negocio del proyecto. En la misma, se muestra en primer lugar al actor Administrador que controla y supervisa las operaciones que se realizan en el sistema de forma general. A partir de ahí, tenemos al rol de Responsable del museo que compone y configura el sitio web que posteriormente será accesible al público general de Internet, mediante un dispositivo móvil o incluso a otras aplicaciones (desarrolladas en otros lenguajes de forma ajena al sistema) mediante servicios web, RSS o XML.

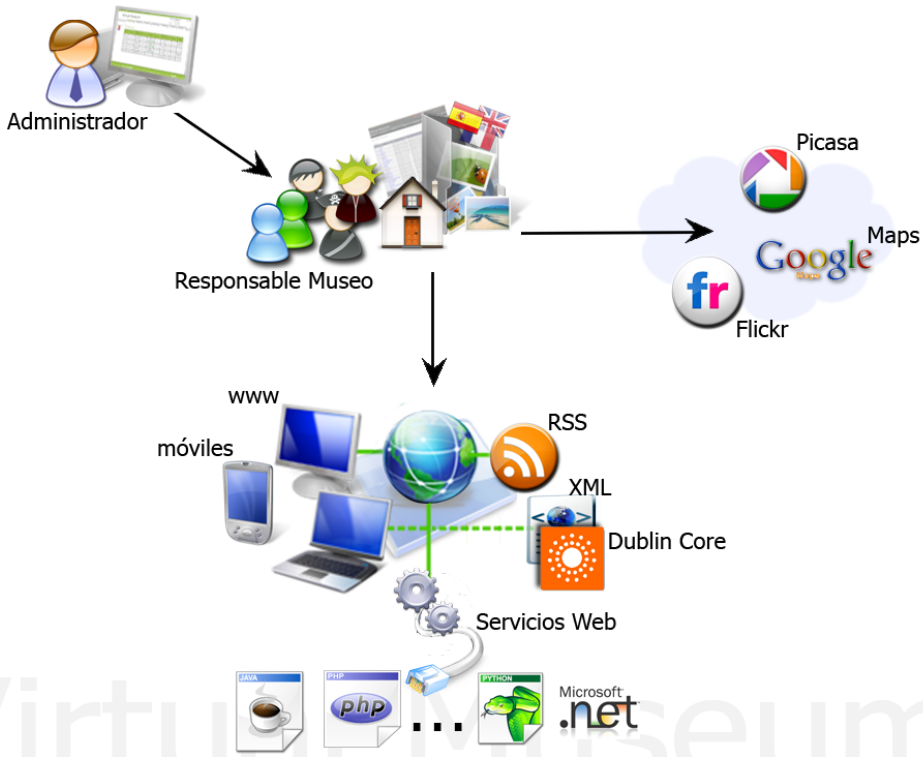


Figura 5.1: Modelo de negocio

La aplicación del Responsable del museo presentará diversas funcionalidades en cuanto a contenidos, idiomas, personalización de apariencia, y además, aportará funciones de comunicación con otros sistemas de Internet como Flickr, Picasa o Google Maps.

En resumen, el proyecto a desarrollar cuenta con diversas funciones y servicios que lo hará útil a los usuarios que deseen abrir a Internet las colecciones de obras u objetos en general.

5.2.4. Características del usuario

Los usuarios de este sistema serán personas familiarizadas al menos con la tecnología web. Estos usuarios tienen conocimientos sobre los sistemas gestores de contenidos y la necesidad del uso de estándares de metadatos para el catalogado de los recursos de una colección.

5.2.5. Restricciones

Para las aplicaciones del administrador y responsable del museo, el sistema debe proporcionar una forma segura de autenticación de usuarios, y autorización de las acciones para las que tienen permiso. Las vistas tanto web como móvil del museo virtual de cara al visitante serán de acceso público, desde el momento en que el responsable del museo las publique.

5.3. Requisitos específicos de la aplicación del administrador

5.3.1. Funcionalidad de la aplicación del administrador

1. La entrada a la aplicación para el administrador deberá ser permitida únicamente a los usuarios que posean una cuenta de administrador. Al menos existirá una cuenta de administrador en el sistema. Un administrador puede cambiar su clave en cualquier momento.
2. El administrador puede crear nuevas cuentas de *Responsable del museo* o borrar cuentas ya existentes. Asimismo, también será capaz de borrar el museo virtual creado por un usuario responsable del museo. La creación de cuentas puede incluir un mecanismo de generación aleatoria de contraseñas. El borrado de una cuenta de responsable del museo conlleva el borrado del museo virtual que hubiera creado. A la inversa, el borrado de un museo virtual no implica el borrado de la cuenta de responsable del museo.
3. Las cuentas de responsable del museo constan de un identificador de cuenta (*login*) y contraseña que se suministran en la aplicación para permitir el acceso. Además, como información adicional, en la creación de una cuenta de director se debe suministrar al menos su nombre, dni, dirección de correo electrónico, el nombre del museo para el cual se va a crear un museo virtual y su dirección física.
4. Como la cuenta de administrador tiene más privilegios con las cuentas de responsable del museo, un administrador puede modificar un museo virtual desde su aplicación de la misma forma en que lo haría el responsable del museo. De la misma forma, el administrador puede cambiar la contraseña de un responsable del museo lo que serviría por ejemplo, para evitar el acceso a la aplicación para el director del museo para ese responsable del museo en particular.
5. El administrador puede importar nuevas plantillas al sistema, ofreciendo así un mecanismo de personalización de la apariencia para los museos generados.
6. El sistema puede recoger estadísticas acerca del uso de las cuentas de responsable del museo, museos virtuales creados, espacio en disco del servidor ocupado, etc. que servirán para dar información al administrador.

5.3.2. Requisitos adicionales de la aplicación del administrador

Además de estas funcionalidades el sistema debe proporcionar al administrador un soporte de ayuda que pueda ser consultado en cualquier momento.

Sería deseable por otra parte que el sistema permitiese una vía de comunicación con el responsable del museo para que éste pueda aportar comentarios.

El sistema debe ofrecer un mecanismo para soportar múltiples idiomas en la visualización de la aplicación.

5.4. Requisitos específicos de la aplicación del director de museo

5.4.1. Funcionalidad de la aplicación del director de museo

1. Un responsable del museo tiene una cuenta en el sistema que le permite la entrada a la creación, modificación y mantenimiento de su museo virtual. Referente a su cuenta, un responsable del museo no podrá cambiar su identificador de cuenta. Esto no implica que en la creación del museo virtual se establezca un título y una ubicación física distinta de la especificada inicialmente. Inicialmente en el sistema no existirá ninguna cuenta de responsable del museo; la creación de estas cuentas es labor del administrador, aunque adicionalmente se podrá habilitar una zona de registro por parte del usuario (teniendo en cuenta algún criterio para evitar *spam* y registros automáticos)

2. A cada cuenta de responsable del museo se le permite la creación de un museo virtual. El responsable del museo introduce los datos relativos a su museo y los contenidos que en él se exponen (*objetos digitales*) por medio de formularios en la aplicación.

3. Cada museo virtual generado puede ser configurado para visualizarse en varios idiomas, aunque será labor del responsable del museo editar los textos de los contenidos para cada uno de los idiomas que configure.

4. El esquema de toda la información introducida se ajustará a un estándar de metadatos preestablecido, con el fin de proporcionar interoperabilidad con otros sistemas existentes. Al seguir un estándar de metadatos, el sistema podrá exportar datos de su museo en parte o en su totalidad a petición del responsable del museo o de otro sistema ajeno.

5. Una vez introducido un conjunto mínimo de datos el responsable del museo puede solicitar un vista previa o publicar su museo virtual. El conjunto mínimo de datos necesarios para publicar o solicitar una vista previa son el nombre del museo, información acerca de un objeto digital y un recorrido conteniendo al menos un objeto digital.

6. Al hacer público un museo virtual el sistema pone a disposición de los visitantes la vista web del museo virtual recién creado. El responsable del museo cuenta con la posibilidad de generar además una vista para móviles y PDA's del museo virtual. Como mínimo, la publicación de un museo virtual debería contar con el nombre del museo, su ubicación física e información acerca de al menos uno de sus objetos.

7. Se pueden asociar ficheros multimedia a los contenidos. Para ello, la aplicación cuenta con un gestor de archivos que permita el envío de estos ficheros multimedia al servidor. Deberá quedar claro que el sistema queda exento de cualquier responsabilidad por la naturaleza u opinión reflejada en la información introducida y por derechos de copyright acerca de los ficheros multimedia.

8. La apariencia del sitio web para el museo virtual generado debe poder personalizable mediante el uso de plantillas.

9. Los contenidos del sitio web generado pueden ser accesibles a través de menús configurados por el propio responsable del museo.

10. La creación o modificación de un museo virtual puede añadir un historial de creación/modificación de forma que se sepa en todo momento qué etapas se han completado y cuándo.

11. Tanto la información del museo introducida como los ficheros multimedia estarán alojados bajo el control del sistema, ofreciendo así un servicio de *hosting*.

12. A partir de los objetos multimedia subidos, aquéllos que sean imágenes se podrán publicar desde la aplicación en Flickr y Picasa.

13. El sistema permitirá la conexión con Google Maps, permitiendo indicar las coordenadas de posición del museo virtual, y dejando al sistema que represente un mapa con dicha localización.

14. El sistema puede recoger estadísticas acerca de los visitantes del museo virtual que servirán de información al responsable del museo.

5.4.2. Requisitos adicionales de la aplicación del responsable del museo

Además de estas funcionalidades el sistema debe proporcionar al responsable del museo un soporte de ayuda que pueda ser consultado en cualquier momento.

Sería deseable por otra parte que el sistema permitiese una vía de comunicación con el visitante para que éste pueda aportar comentarios.

El sistema debe ofrecer un mecanismo para soportar múltiples idiomas en la visualización de la aplicación.

5.5. Requisitos específicos de la aplicación del visitante

5.5.1. Funcionalidad de la aplicación del visitante

1. De cara al visitante el sistema permite la visualización de los museos virtuales creados por los directores de museos. Así, esta aplicación únicamente proporciona una vista web y una vista para móviles o PDA's del museo virtual.

2. Tanto la vista web como la móvil deben ocultar aquellas secciones del museo virtual que no hayan sido cumplimentadas por el responsable del museo.

3. Si el museo virtual está habilitado para ello, puede exponer un servicio web de acceso a porciones de los contenidos del propio museo, permitiendo así la interoperabilidad con otros sistemas.

4. Si el museo virtual está habilitado para ello, puede sindicar los contenidos de actualización reciente en forma de feed RSS, permitiendo así el acceso a los contenidos del museo mediante lectores de feeds o incluso otros sistemas.

5. El museo virtual expondrá la información Dublin Core asociada a sus contenidos, principalmente para aquellos que sean obras.

Capítulo 6

DESCRIPCIÓN DE CASOS DE USO

Si añades un poco a lo poco y lo haces así con frecuencia,
pronto llegará a ser mucho.

Hesíodo

Índice del Capítulo

6.1. Introducción

El esfuerzo principal en la fase de requisitos es desarrollar un modelo del sistema que se va a construir, y la utilización de los casos de uso es una forma adecuada de crear ee modelo. Esto es debido a que los requisitos funcionales se estructuran de forma relativamente natural mediante casos de uso, y a que la mayoría de los otros requisitos no funcionales son específicos de un solo caso de uso, y pueden tratarse en el contexto de ese caso de uso. Los requisitos no funcionales restantes, comunes para muchos casos de uso, se pueden mantener en una sección aparte.

6.2. Casos de Uso para el Administrador

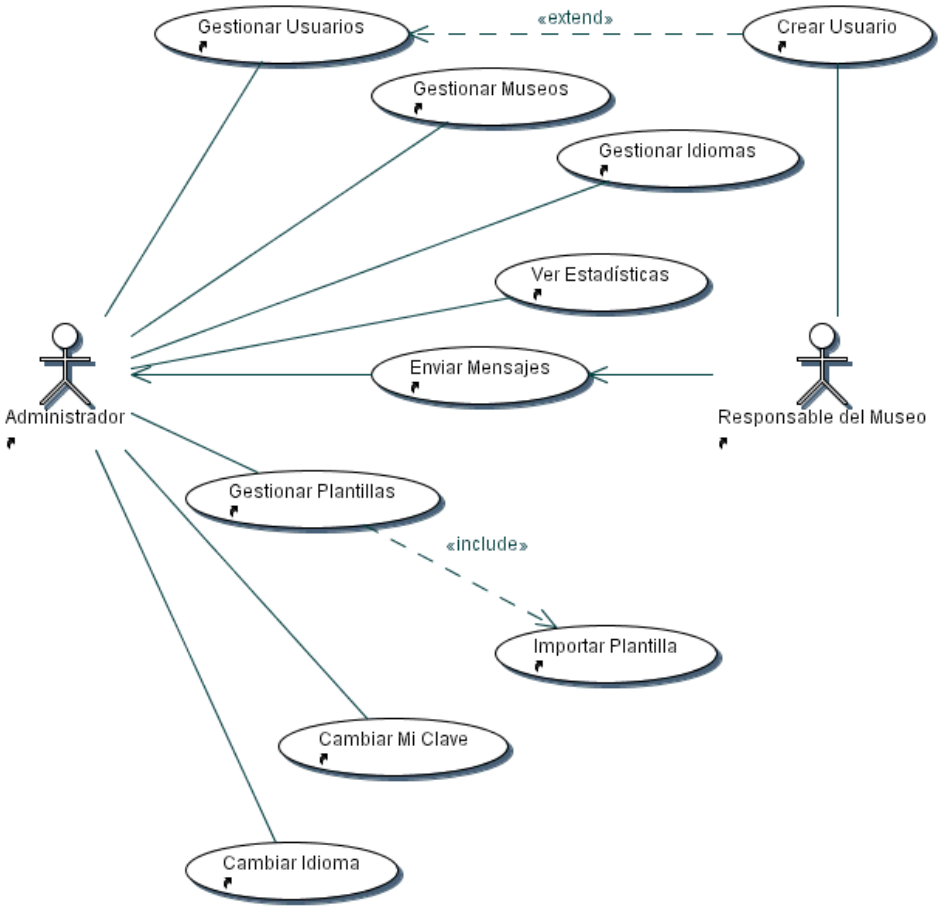


Figura 6.1: Diagrama de Casos de Uso para el administrador

CU.A01 - Gestionar Usuarios		
Descripción	Reúne la funcionalidad para dar de alta, eliminar y modificar usuarios con el rol responsable del museo	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema ofrece una lista de todos los usuarios con el rol responsable del museo
	2	El administrador elige un responsable del museo para editar su información
	3	El sistema muestra la información del usuario permitiendo la modificación de los campos
	4	Al guardar se vuelve al listado de usuarios
Excepciones	**Paso**	**Acción**
	2	El administrador selecciona crear un nuevo responsable del museo; como no hay usuario seleccionado, el sistema mostrará la información del usuario para guardar incluyendo el login del usuario que se está creando.
	2	El administrador escoge uno o más usuarios para eliminarlos, previa confirmación

Cuadro 6.1: CU.A01 - Gestionar Usuarios

CU.A02 - Gestionar Museos		
Descripción	Reúne la funcionalidad para dar de alta, eliminar y modificar museos virtuales	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema ofrece una lista de todos los museos virtuales
	2	El administrador elige un museo para administrar lo que conlleva poder gestionar el museo completamente como si fuese el propio responsable del museo
Excepciones	**Paso**	**Acción**
	2	El administrador selecciona crear un nuevo museo virtual en cuyo caso el sistema mostrará la información del museo en blanco para guardarlo
	2	El administrador escoge uno o más museos para eliminarlos, previa confirmación

Cuadro 6.2: CU.A02 - Gestionar Museos

CU.A03 - Enviar Mensajes		
Descripción	Visualización de los mensajes enviados por los usuarios de los museos virtuales	
Secuencia normal	**Pasos**	**Acción**
	1	El actor responsable del museo envía un mensaje desde su aplicación
	2	El sistema lo recoge informándole de que le llegará al administrador
	3	El sistema ofrece una lista de todos los mensajes enviados por los usuarios de los museos. No se podrá modificar ningún mensaje

Cuadro 6.3: CU.A03 - Enviar Mensajes

CU.A04 - Ver Estadísticas		
Descripción	Visualización de las estadísticas recopiladas por cada museo virtual. Como existirán pantallas dedicadas a la gestión de usuarios y gestión de museos, ésta ofrecerá una vista agrupada de información estadística asociada al museo virtual y su responsable del museo	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema ofrece una lista de todas las estadísticas recopiladas

Cuadro 6.4: CU.A04 - Ver Estadísticas

CU.A05 - Gestionar Plantillas		
Descripción	Reúne la funcionalidad para dar de alta, eliminar y modificar plantillas	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema ofrece una lista de todas las plantillas
	2	El administrador elige una para editar sus detalles
	3	El sistema presenta los detalles de la plantilla junto con su vista previa y el árbol de archivos/directorios que componen la plantilla
	4	El administrador puede modificar el contenido de aquellos archivos que sean texto plano y visualizar el resto
Excepciones	**Paso**	**Acción**
	2	El administrador escoge una o más plantillas para eliminarlos, previa confirmación

Cuadro 6.5: CU.A05 - Gestionar Plantillas

CU.A06 - Importar plantilla		
Descripción	Permite importar un empaquetado de la plantilla conteniendo todos los archivos necesarios para su visualización	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra un diálogo de examinar archivo para seleccionar la plantilla (el archivo empaquetado) de su árbol de directorios local
	2	El administrador selecciona el archivo adecuado
	3	El sistema desempaqueta el archivo ya subido al servidor
	4	El sistema procesa el archivo desempaquetado registrándolo como nueva plantilla para su uso
Excepciones	**Paso**	**Acción**
	4	Si al procesar el archivo subido resulta que no presenta la estructura fijado por las plantillas en Virtual Museum, se desecha como plantilla informando al usuario

Cuadro 6.6: CU.A06 - Importar plantilla

CU.A07 - Gestionar Idiomas		
Descripción	Reúne la funcionalidad para dar de alta, eliminar y modificar idiomas soportados por el sistema global	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema ofrece una lista de todos los idiomas
	2	El administrador elige un idioma para editar su información
	3	El sistema muestra la información del idioma permitiendo la modificación de los campos
	4	Al guardar se vuelve al listado de idiomas
Excepciones	**Paso**	**Acción**
	2	El administrador selecciona crear un idioma; como no hay idioma seleccionado, el sistema mostrará la información del idioma para guardar incluyendo el código del idioma que se está creando.
	2	El administrador escoge uno o más idiomas para eliminarlos, previa confirmación

Cuadro 6.7: CU.A07 - Gestionar Idiomas

CU.A08 - Cambiar Mi Clave		
Descripción	Cambio de la clave del usuario administrador que ha usado para entrar a su aplicación	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra un diálogo para escribir la nueva clave dos veces para evitar confusiones
	2	El administrador introduce su clave por duplicado
	3	El sistema valida la clave introducida
	4	El sistema almacena la clave del usuario administrador para futuras entradas al sistema
Excepciones	**Paso**	**Acción**
	2	El administrador puede solicitar generar una clave aleatoria de forma automática por el sistema
	3	Si al validar la clave por duplicado se han especificado claves distintas, o directamente no se especifica, la operación se cancela

Cuadro 6.8: CU.A08 - Cambiar Mi Clave

CU.A09 - Cambiar Idioma		
Descripción	Cambio del idioma en el que se está presentando la aplicación, afectando a todos los mensajes de la misma	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra un selector de los idiomas que soporta su aplicación
	2	El administrador selecciona uno de los idiomas
	3	El sistema cambia el idioma actual al elegido por el administrador redireccionando a la primera página que ve según entra a la aplicación

Cuadro 6.9: CU.A09 - Cambiar Idioma

6.3. Casos de Uso para el Director de museo

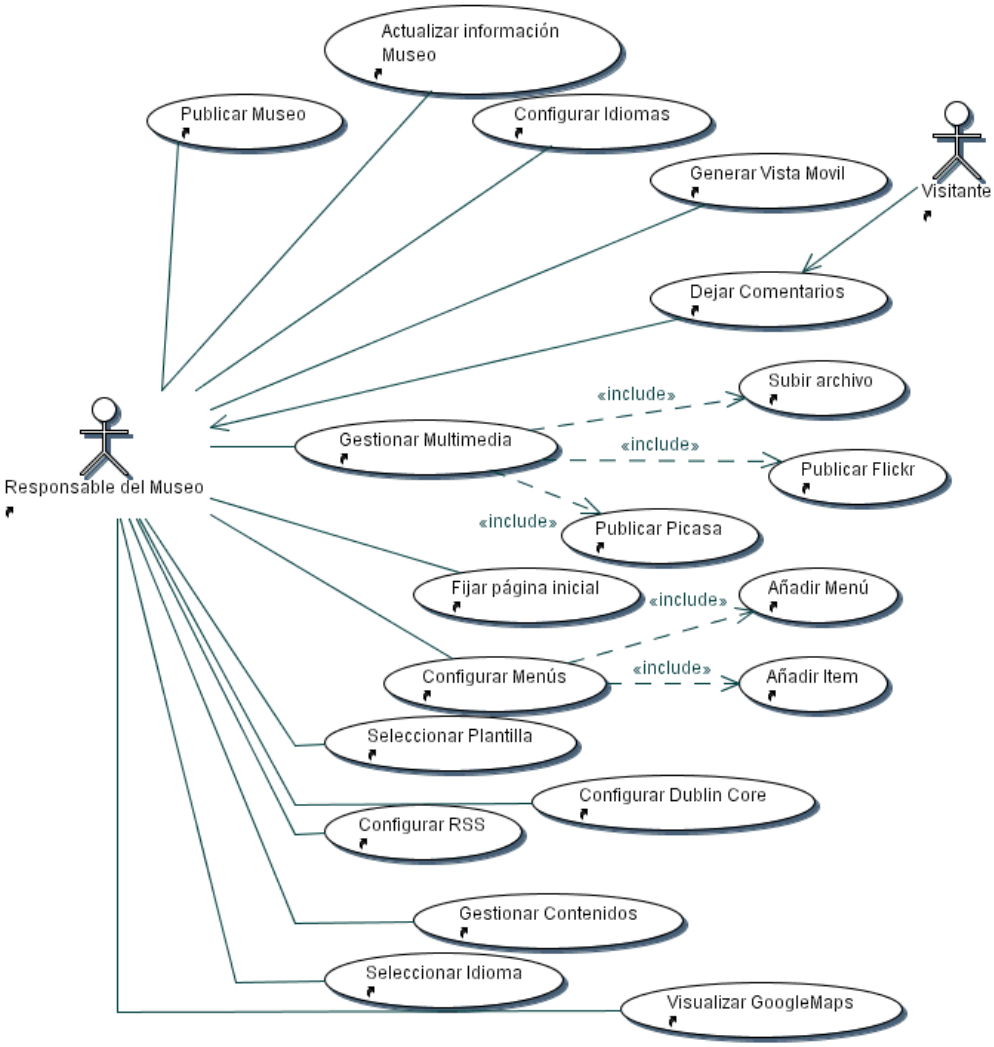


Figura 6.2: Diagrama de Casos de Uso para el responsable del museo

CU.D01 - Actualizar información Museo		
Descripción	Acceso a la información interna del museo, que se usó para crear el museo virtual.	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra la información actual del museo permitiendo al usuario la modificación de cualquiera de sus campos
	2	El responsable del museo actualiza la información
	3	El sistema guarda la información editada, teniendo en cuenta que, como se incluye el propio nombre del museo, hará falta recargar el nombre mostrado

Cuadro 6.10: CU.D01 - Actualizar información Museo

CU.D02 - Publicar Museo		
Descripción	Publicación/despublicación del museo para hacerlo visible vía web al visitante	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra el estado actual del museo: publicado o no publicado (”despublicado”)
	2	El responsable del museo elige si desea publicar o no el museo
	3	El sistema actualiza en consonancia el estado del museo

Cuadro 6.11: CU.D02 - Publicar Museo

CU.D03 - Generar Vista Móvil		
Descripción	Generación de una vista para dispositivos móviles a partir de un recorrido existente y uno de los idiomas del museo	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra la lista de recorridos para el móvil existentes y la lista de idiomas en las que está configurado el museo virtual. Además, muestra la lista de vistas móviles que se han generado previamente
	2	El responsable del museo elige el recorrido y el idioma para los que desea generar la vista móvil
	3	El sistema genera la vista como un archivo *jar*, actualizando su fecha de generación y su ruta
	4	El sistema le informa al responsable del museo de la generación de la vista permitiendo su descarga de la misma forma que para las vistas ya generadas anteriormente

Cuadro 6.12: CU.D03 - Generar Vista Móvil

CU.D04 - Configurar Idiomas		
Descripción	Configuración del idioma principal y complementarios para los que que estará disponible el museo virtual generado y se editarán sus contenidos	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra los idiomas configurados actualmente, resaltando el idioma principal o predeterminado
	2	El sistema muestra la lista de idiomas disponibles globalmente
	3	El responsable del museo selecciona un idioma de entre los disponibles para añadirlo a su museo
	4	El sistema guarda el nuevo idioma añadido, mostrándolo en la lista de idiomas configurados actualmente como uno complementario
Excepciones	**Paso**	**Acción**
	3	El responsable del museo cambia el idioma predeterminado a otro de los configurados actualmente
	3	El responsable del museo escoge uno o más idiomas para eliminarlos, previa confirmación. El idioma predeterminado no se puede eliminar
	4	Si el museo todavía no tenía configurado ningún idioma, el idioma recién añadido aparece como idioma predeterminado

Cuadro 6.13: CU.D04 - Configurar Idiomas

CU.D05 - Dejar Comentarios		
Descripción	Brinda una vía de comunicación entre el visitante y el responsable del museo para transmitirle comentarios acerca del museo virtual	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra los comentarios enviados con anterioridad
	2	El visitante aporta una opinión o sugerencia sobre aquellos aspectos y servicios del museo virtual que podría contemplar
	3	El sistema guarda el comentario para que lo pueda consultar posteriormente el responsable del museo

Cuadro 6.14: CU.D05 - Dejar Comentarios

CU.D06 - Seleccionar Idioma		
Descripción	Cambio del idioma en el que se está presentando la aplicación, afectando a todos los mensajes de la misma	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra un selector de los idiomas que soporta su aplicación
	2	El responsable del museo selecciona uno de los idiomas
	3	El sistema cambia el idioma actual al elegido por el responsable del museo redireccionando a la primera página que ve según entra a la aplicación

Cuadro 6.15: CU.D06 - Seleccionar Idioma

CU.D07 - Gestionar Multimedia		
Descripción	Reúne la funcionalidad que permite al responsable del museo administrar sus objetos multimedia	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra la lista de objetos multimedia pertenecientes al responsable del museo. Aquellos ficheros que presenten el tipo MIME css se pueden editar, y el resto únicamente visualizar
	2	Si el responsable del museo desea subir un archivo nuevo se realiza el caso de uso CU.D08
	3	Si el responsable del museo desea publicar una imagen existente en Picasa se realiza el caso de uso CU.D17
	4	Si el responsable del museo desea publicar una imagen existente en Flickr se realiza el caso de uso CU.D18
	5	El responsable del museo puede escoger uno o más archivos para eliminarlos, previa confirmación.

Cuadro 6.16: CU.D07 - Gestionar Multimedia

CU.D08 - Subir archivo		
Descripción	Permite subir nuevos archivos al gestor multimedia	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra un diálogo de examinar archivo para seleccionar el objeto multimedia a subir de su árbol de directorios local
	2	El responsable del museo selecciona un archivo
	3	El sistema sube el archivo al servidor
	4	El sistema procesa el archivo registrándolo como un nuevo objeto multimedia analizando su tipo MIME para permitir diferentes operaciones con él posteriormente

Cuadro 6.17: CU.D08 - Subir archivo

CU.D09 - Fijar página inicial		
Descripción	Fija la página inicial que se mostrará por defecto al comenzar la visita a un museo virtual	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra la lista de contenidos existentes para el museo virtual
	2	El responsable del museo selecciona uno de los contenidos para que sea el primero que se muestra cuando un visitante entra a un museo
	3	El sistema actualiza la selección como contenido para la página inicial
Secuencia normal	**Pasos**	**Acción**
	1	Si el museo todavía no tiene contenidos, el sistema informe del hecho y no permite fijar nada

Cuadro 6.18: CU.D09 - Fijar página inicial

CU.D10 - Configurar Menús		
Descripción	Reúne la funcionalidad de la configuración de menús que se visualizarán en el sitio web para el museo virtual generado	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra la lista de menús configurados previamente para el museo virtual
	2	Si el responsable del museo desea añadir un menú nuevo, se realiza el caso de uso CU.D11
	3	El responsable del museo escoge un menú existente para ver sus detalles
	4	El sistema muestra la lista de items para el menú seleccionado, y los detalles del menú permitiendo su modificación
	5	Si el responsable del museo desea añadir un item nuevo al menú, se realiza el caso de uso CU.D12
	6	El responsable del museo puede reordenar los items para que el sitio web del museo los muestre en el orden especificado
	7	El sistema actualiza toda la información del menú y vuelve a la lista de menús configurados
Secuencia normal	**Pasos**	**Acción**
	2	El responsable del museo escoge uno o más menús para eliminarlos, previa confirmación
	4	El responsable del museo escoge uno o más items del menú para eliminarlos, previa confirmación

Cuadro 6.19: CU.D10 - Configurar Menús

CU.D11 - Añadir Menú		
Descripción	Permite añadir un menú nuevo	
Secuencia normal	**Pasos**	**Acción**
	1	El responsable del museo indica el título del menú a añadir, así como su estilo CSS y el título del menú para hacerlo corresponder con el de la plantilla
	2	El sistema almacena el menú

Cuadro 6.20: CU.D11 - Añadir Menú

CU.D12 - Añadir Item		
Descripción	Permite añadir un nuevo item a un menú existente	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra la lista de todos los contenidos y las categorías para el museo virtual
	1	El responsable del museo selecciona o un contenido o una de las categorías para añadirlo como un nuevo item al menú
	2	El sistema almacena el nuevo item, ocupando la última posición dentro del menú

Cuadro 6.21: CU.D12 - Añadir Item

CU.D13 - Seleccionar Plantilla		
Descripción	Permite al responsable del museo escoger una plantilla para personalizar la apariencia del sitio web para el museo virtual generado	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra la lista de todas las plantillas disponibles globalmente, así como la plantilla que tiene el museo actualmente
	1	El responsable del museo selecciona una de las plantillas para cambiarla por la actual
	2	El sistema almacena la plantilla seleccionada, de forma que las posteriores visitas al sitio web del museo virtual generado se mostrarán usando dicha plantilla

Cuadro 6.22: CU.D13 - Seleccionar Plantilla

CU.D14 - Configurar Dublin Core		
Descripción	Permite configurar las opciones relacionadas con Dublin Core: incrustar metadatos DC en el sitio web para el museo virtual generado y publicar servicio web de comunicación con otras aplicaciones	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra si están activos el servicio de publicación del servicio web y la inclusión de metadatos Dublin Core en el sitio web para el museo virtual
	2	El responsable del museo activa o desactiva dichos servicios a voluntad
	3	El sistema actualiza la nueva configuración

Cuadro 6.23: CU.D14 - Configurar Dublin Core

CU.D15 - Configurar RSS		
Descripción	Permite configurar la activación del servicio de publicación de feeds RSS con los contenidos actualizados durante el último mes	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra si está activo o no el servicio de publicación de feeds RSS para el museo virtual
	2	El sistema muestra una vista previa con la lista de los contenidos que se van a incluir en el feed, a saber, contenidos que se han actualizado durante el último mes
	3	El responsable del museo activa o desactiva dicho servicio a voluntad
	4	El sistema actualiza la nueva configuración

Cuadro 6.24: CU.D15 - Configurar RSS

CU.D16 - Gestionar Contenidos		
Descripción	Reúne la funcionalidad para dar de alta, eliminar y modificar los contenidos del museo virtual	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema ofrece una lista de todos los contenidos
	2	El administrador elige un contenido para editar su título y texto, y los campos adicionales de cada tipo de contenido, permitiendo la modificación de los campos
	3	Al guardar se vuelve al listado de contenidos
Excepciones	**Paso**	**Acción**
	2	Si el museo virtual está configurado en varios idiomas, se presenta el par título/texto para cada idioma
	2	El responsable del museo escoge uno o más contenidos para eliminarlos, previa confirmación

Cuadro 6.25: CU.D16 - Gestionar Contenidos

CU.D17 - Publicar Picasa		
Descripción	Permite subir imágenes propias el museo virtual a Picasa	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra desde el listado de archivos multimedia un enlace para publicar imágenes a Picasa
	2	El responsable del museo selecciona una imagen para publicar
	3	El sistema se conecta con Picasa para enviar la imagen
	4	Como la imágenes también se podrían eliminar desde Picasa, se puede refrescar el estado de imágenes para comprobar las imágenes que se tienen actualmente publicadas en Picasa

Cuadro 6.26: CU.D17 - Publicar Picasa

CU.D18 - Publicar Flickr		
Descripción	Permite subir imágenes propias el museo virtual a Flickr	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema muestra desde el listado de archivos multimedia un enlace para publicar imágenes a Flickr
	2	El responsable del museo selecciona una imagen para publicar
	3	El sistema envía un correo a Flickr adjuntando la imagen que se desea publicar
	4	El responsable del museo puede refrescar el estado de imágenes enviadas a Flickr, para verificar si ya está subida
	5	Como la imágenes también se podrían eliminar desde Flickr, se puede refrescar el estado de imágenes para comprobar las imágenes que se tienen actualmente publicadas en Flickr

Cuadro 6.27: CU.D18 - Publicar Flickr

CU.D19 - Visualizar GoogleMaps		
Descripción	Permite visualizar la información geográfica del museo especificada mediante GoogleMaps	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema ofrece un formulario para completar la información geográfica asociada al museo virtual
	2	El responsable del museo rellena dicha información (longitud, latitud, altura)
	3	El sistema la actualiza y ofrece una visualización de dicha posición mediante GoogleMaps (bien como Javascript o como una imagen simple)

Cuadro 6.28: CU.D19 - Visualizar GoogleMaps

6.4. Casos de Uso para el Visitante

El apartado de casos de uso para el visitante se plantea de forma genérica para las funcionalidades de la vista del museo virtual generado ya que la interacción estará condicionada por el diseño de la plantilla en el caso de la vista web y el esquema de pantallas de la aplicación base de la vista móvil.

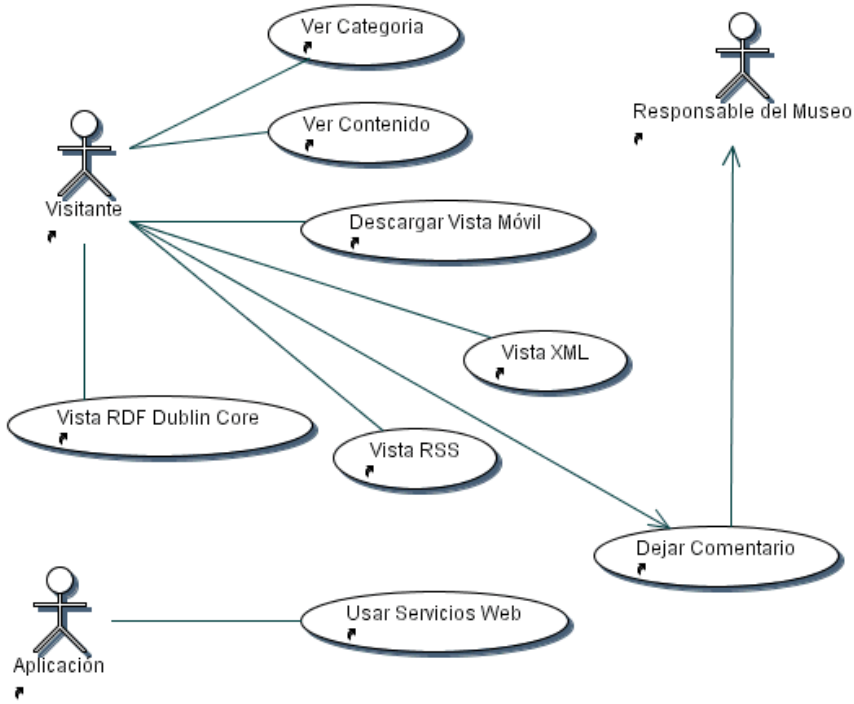


Figura 6.3: Diagrama de Casos de Uso para el visitante

CU.V01 - Descargar Vista Móvil		
Descripción	Permite descargarse al visitante una vista móvil configurada por el responsable del museo	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema según la plantilla escogida, y si se ha decidido mostrarla, muestra una serie de enlaces para descargarse las aplicaciones para dispositivos móviles
	2	El visitante escoge descargarse alguna de ellas
	3	El visitante la vuelca sobre el dispositivo móvil. No se establecen las vías para transferir la aplicación al dispositivo móvil
	4	La aplicación *standalone* se ejecuta como una aplicación más en el dispositivo móvil

Cuadro 6.29: CU.V01 - Descargar Vista Móvil

CU.V02 - Vista XML		
Descripción	Permite visualizar una obra como XML en vez de HTML	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema ofrece un enlace similar al de ver un contenido concreto para visualizar una obra existente como XML, siguiendo Dublin Core
	2	El navegador *renderiza* la información XML en la forma como esté configurado para ello

Cuadro 6.30: CU.V02 - Vista XML

CU.V03 - Vista RSS		
Descripción	Permite visualizar una obra como RSS en vez de HTML	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema según la plantilla escogida, y si se ha decidido mostrarla, muestra un enlace de sindicación a los contenidos RSS habilitados
	2	El visitante puede visualizarlos directamente en su navegador (al tratarse de XML) o bien utilizar un lector de feeds para una lectura más cómoda

Cuadro 6.31: CU.V03 - Vista RSS

CU.V04 - Ver metas Dublin Core		
Descripción	Incrusta los términos DC configurados por el responsable del museo dentro del html *renderizado* para una obra	
Secuencia normal	**Pasos**	**Acción**
	1	El sistema según la plantilla escogida, y si se ha decidido mostrarla, añade una serie de etiquetas *meta* pertenecientes a cada uno de los términos Dublin Core que el responsable del museo ha configurado previamente
	2	El visitante puede visualizarlos directamente en su navegador (a través del código fuente de la página) o bien utilizar alguna aplicación adicional o *plugin* añadido al navegador

Cuadro 6.32: CU.V04 - Ver metas Dublin Core

CU.V05 - Usar servicios web		
Descripción	El sitio web generado para un museo virtual puede publicar un servicio web para el acceso (restringido) a un subconjunto de los atributos de las obras	
Secuencia normal	**Pasos**	**Acción**
	1	Si el responsable del museo lo ha habilitado, el sistema publica un servicio web de acceso a los textos del contenido indicado
	2	Una aplicación ajena puede acceder a la información ofrecida por dicho servicio, de forma remota y transparente al lenguaje usado

Cuadro 6.33: CU.V05 - Usar servicios web

CU.V06 - Ver Contenido		
Descripción	El sitio web generado para un museo virtual *renderiza* los atributos asociados a un contenido concreto enmarcado según el diseño de la plantilla seleccionada	
Secuencia normal	**Pasos** 1 2	**Acción** El visitante selecciona visitar un contenido mediante su dirección (a la que accedió por el portal Virtual Museum, por un item del menú del propio sitio, búsqueda, etc) El sitio web generado muestra los atributos asociados (normalmente, como mínimo el título y el texto del contenido) al contenido.
Excepciones	**Paso** 1	**Acción** Si el museo virtual tiene un contenido por defecto y el visitante al entrar por primera vez al museo virtual no especifica un contenido concreto, el sistema muestra dicho contenido por defecto (o de portada)

Cuadro 6.34: CU.V06 - Ver Contenido

CU.V07 - Ver Categoría		
Descripción	El sitio web generado para un museo virtual *renderiza* los atributos asociados a los contenidos pertenecientes a una categoría dada enmarcados según el diseño de la plantilla seleccionada	
Secuencia normal	**Pasos** 1 2	**Acción** El visitante selecciona visitar una categoría mediante su dirección (a la que accedió por el portal Virtual Museum, por un item del menú del propio sitio, búsqueda, etc) El sitio web generado muestra el conjunto de contenidos existentes para la categoría dentro del museo virtual dado. La plantilla podrá especificar distintas formas de visualización

Cuadro 6.35: CU.V07 - Ver Categoría

Capítulo 7

ANÁLISIS

Existe cierta majestuosidad en la simplicidad, lo cual está muy por encima de la curiosidad del ingenio.

Alexander Pope

Índice del Capítulo

7.1. Introducción

Durante el análisis, se analizan los requisitos que se han descrito en los dos capítulos anteriores, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea más fácil de mantener y que ayude a estructurar el sistema entero, incluyendo la arquitectura.

7.2. Diagrama de Clases de Análisis

Los diagramas de clases de análisis mostrados a continuación presenta una visión más precisa y utilizando un lenguaje más cercano a los desarrolladores. Se debe indicar que la descripción de los atributos del modelo del sistema se pueden ver posteriormente en el diagrama de clases de diseño y en los modelos del CD anexo.

7.2.1. Administrador

A partir de la figura 7.1 se pueden ver las relaciones entre las clases de entidad, interfaz y lógica presentes desde el punto de vista del actor Administrador:

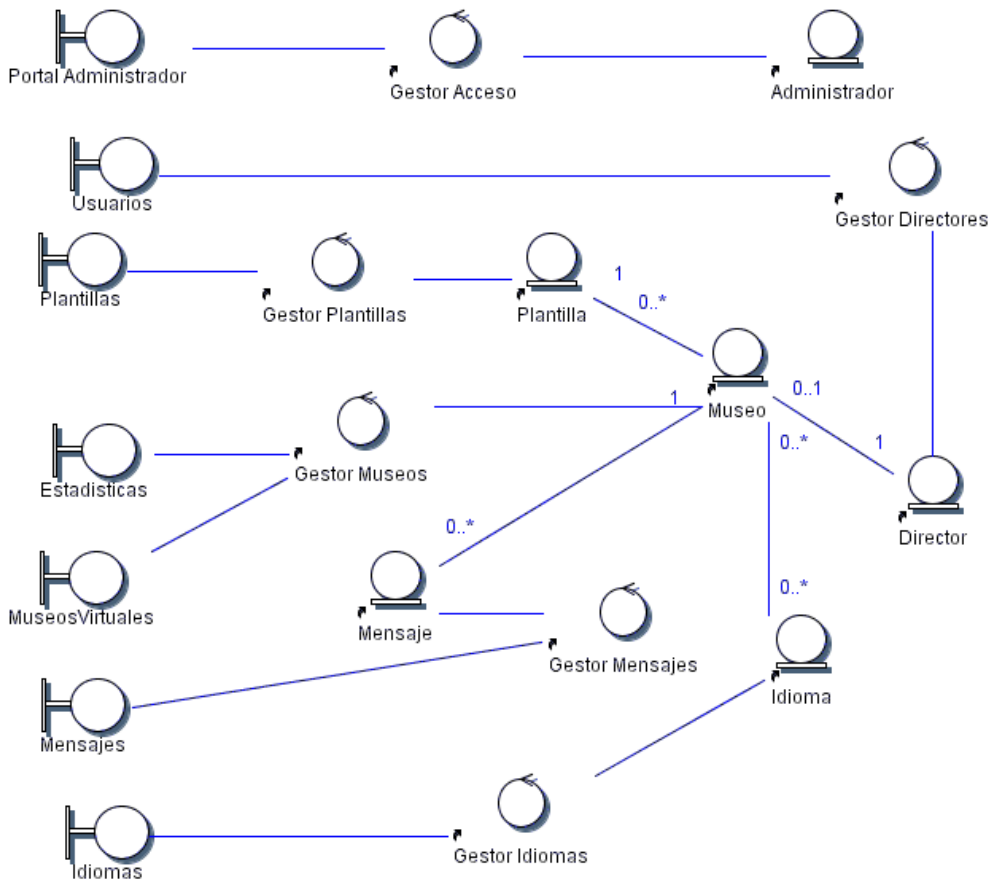


Figura 7.1: Diagrama de Clases de Análisis relativas al Administrador

7.2.2. Director

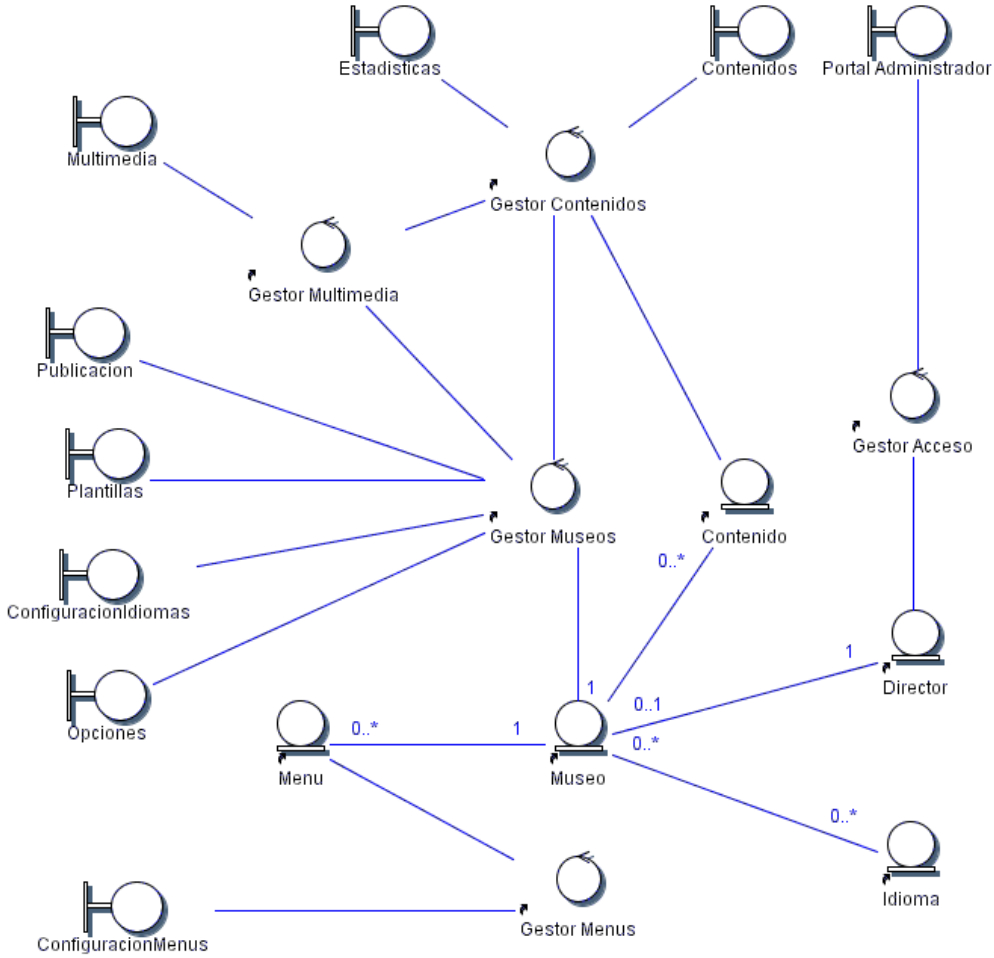


Figura 7.2: Diagrama de Clases de Análisis relativas al Responsable del museo

7.3. Realización de casos de uso-análisis

Las realizaciones de casos de uso-análisis se presentan en forma de diagramas de interacción, y más concretamente como diagramas de colaboración. Para su visualización puede dirigirse al CD anexo y abrir el modelo de análisis dentro del directorio ”together”.

Capítulo 8

DESCRIPCIÓN DE LA ARQUITECTURA

He hecho esta carta más larga de lo usual
porque no tengo tiempo para hacer una más corta.

Blaise Pascal

Índice del Capítulo

8.1. Introducción

A grandes rasgos, uno de los objetivos del diseño es adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, tecnologías de distribución, etc. En el diseño se modela el sistema y se encuentra su forma y arquitectura para que soporte todos los requisitos. Entre el conjunto de requisitos obtenidos se incluyen las siguientes restricciones de diseño de las que parte el proyecto:

- Desarrollo del sistema siguiendo la plataforma Java EE, que sea escalable en cuanto a tecnologías empleadas.
- Las aplicaciones fruto del desarrollo son web, accesibles vía internet en general.
- Uso de soluciones software de libre distribución.

A lo largo de este capítulo se presentarán las distintas tecnologías elegidas para el desarrollo del sistema y cómo se relacionan para la elaboración de la arquitectura del sistema. Una vez expuesta la arquitectura compuesta por los distintos subsistemas, interfaces y dependencias, en el siguiente capítulo se describirá el modelo de clases de diseño que se utilizará como punto de partida fundamental para la fase de implementación.

8.2. Arquitectura Java EE

Java 2 Platform Enterprise Edition o Java EE es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java con **arquitectura distribuida de n niveles**, basándose ampliamente en componentes de software modulares ejecutándose típicamente sobre un servidor de aplicaciones. La plataforma Java EE está definida por una especificación, similar a otras especificaciones del JCP (*Java Community Process*); Java EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son conformes a Java EE; no obstante sin un estándar de ISO o ECMA.

La plataforma Java EE se podría ver como la agregación de un conjunto de especificaciones, un test de compatibilidad (*Java EE Compatibility Test Suite* - CTS), la implementación de referencia y un conjunto de guías de desarrollo y de prácticas aconsejadas denominadas *Java EE BluePrints*.

En cuanto a las tecnologías que soportan este desarrollo basado en varias capas podemos hacer una división en tres categorias: tecnologías de componentes, servicios y comunición. Las tecnologías de componentes son aquéllas usadas por los desarrolladores para crear las partes esenciales de la aplicación, a saber, la interfaz de usuario y la lógica de negocio. Un componente es una unidad software a nivel de la aplicación. Además de los componentes JavaBeans que son parte de la plataforma J2SE, la plataforma Java EE soporte otros tipos de componentes: applets, clientes de aplicación, EJBs, componentes web y adaptadores de recursos. Todos los componentes Java EE necesitan soporte en tiempo de ejecución de una entidad a nivel de sistema llamada contenedor. Un servidor de

aplicaciones Java EE proporciona contenedores para EJBs y para componentes web. Los applets y los clientes de aplicación se ejecutan del lado del cliente, mientras que los EJBs, los componentes web y adaptadores de recursos se ejecutan del lado del servidor. Excepto los adaptadores de recursos, los desarrolladores, frecuentemente, diseñan e implementan los componentes de la aplicación Java EE. Por su parte, los proveedores de los sistemas de información son los que normalmente nos darán los adaptores de recursos.

Los contenedores son entornos de ejecución estandarizados que proveen servicios especificos a los componentes. Los componentes de una aplicación esperarán por tanto que estos servicios estén disponibles en cualquer plataforma Java EE de cualquier distribución. Por ejemplo, todos los contenedores web Java EE proveen soporte en tiempo de ejecución para responder a una petición de cliente (*request*), realizar algún tipo de procesamiento en tiempo de la petición (como la invocación de páginas JSP o de algún servlet), y devolver los resultados al cliente. Además, brindan APIs para dar soporte a la gestión de sesiones. Todos los contenedores EJB dan soporte automático para la gestión de transacciones y el ciclo de vida de los EJBs, así como la búsqueda de beans. Los contenedores también proveen de acceso estandarizado a los sistemas de información; por ejemplo, la API JDBC da acceso a bases de datos relacionales.

Los componentes web están alojados en un contenedor web. Dicho contenedor tiene entre sus funciones dar los servicios de red (soportando el protocolo HTTP y a veces HTTPS) a través de los cuales las peticiones y respuestas son enviadas, decodifica las peticiones y formatea las respuestas. Un contenedor EJB por su parte, suministra servicios de transacciones, persistencia y acceso a los servicios Java EE y APIs de comunicación. En la figura 8.1 puede verse la organización y relaciones entre componentes y contenedores.

Adicionalmente, los contenedores facilitan un mecanismo para elegir el comportamiento de la aplicación en tiempo de ensamblado o despliegue. A través del uso de **descriptores de despliegue** (ficheros XML que especifican el comportamiento del componente o contenedor), se pueden configurar los componentes en función del entorno del contenedor específico en el que se van a desplegar, más que hacerlo en el propio código del componente. Este comportamiento configurable abarca características entre las que se incluyen seguridad, control de transacciones y otras responsabilidades de gestión.

Dentro de las prácticas recomendadas o **Java EE BluePrints**, la plataforma Java EE define un modelo de desarrollo encaminado a la creación de aplicaciones basadas en n capas. Las capas se suelen organizar para que unas se apoyen en otras. Esta separación por capas en un sistema presenta ventajas muy importantes. La primera de ellas es que existe poco acoplamiento entre las mismas, de modo que es mucho más fácil hacer modificaciones en ellas sin que interfieran con las demás. Todo esto redunda en la obtención de mejoras en cuanto a mantenibilidad, escalabilidad y reutilización de componentes. Otra de las ventajas que se obtienen es la posibilidad de intercambio de los componentes usados en las distintas capas, sin que ello interfiera al resto de las capas. Las prácticas recomendadas de Java EE proponen una arquitectura basada en una capa de interfaz de usuario, una o más capas intermedias que proveen de servicios al usuario y lógica de negocio para la aplicación, y una capa de acceso a los sistemas de información.

La figura 8.2 muestra lo que sería una posible arquitectura Java EE. El modelo que aparece en la figura está dividido en varias capas, con una separación clara entre presentación, lógica de negocio y sistemas de información. En este caso además, podemos ver como se trata de un modelo mixto. En la parte de arriba tenemos que se recorre un camino por una estructura de tres capas (aplicación - lógica

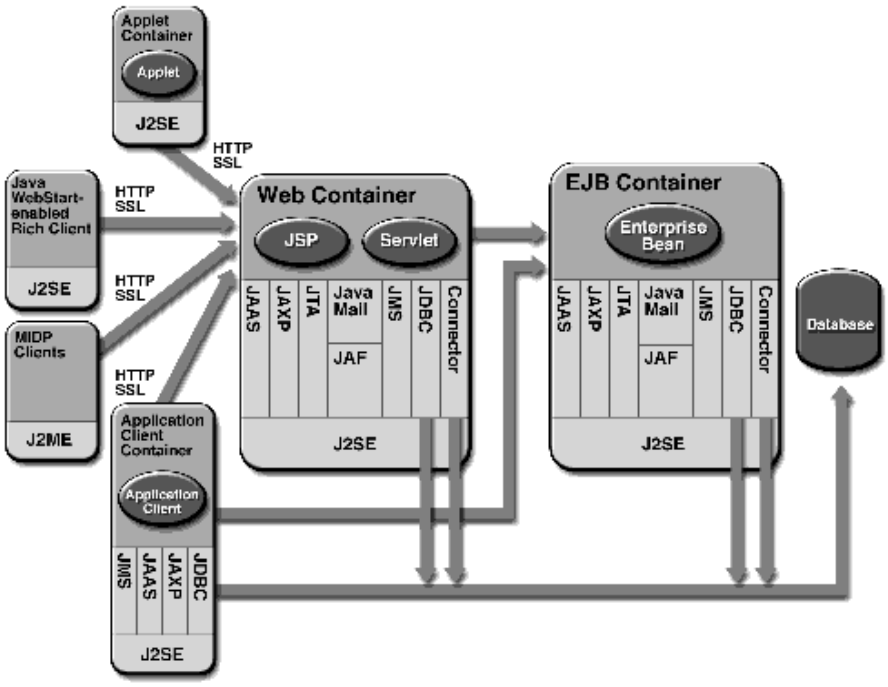


Figura 8.1: Componentes y contenedores en la plataforma Java EE

de negocio - sistemas de información), mientras que por el segundo camino recorremos una estructura de cuatro capas (aplicación - lógica de presentación - lógica de negocio - sistemas de información).

Por otra parte, la capa Java EE de cliente soporta diversos tipos de clientes que interactúan con los componentes del lado del servidor. Entre ellos podemos encontrar los navegadores web (que usan páginas HTML estáticas, páginas HTML generadas dinámicamente por JSPs, o Java applets), aplicaciones Java de escritorio (*standalone*), clientes ricos basados en *Java Web Start*, o clientes wireless basados en MIDP (*Mobile Information Device Profile*) que dan un entorno J2ME completo para los dispositivos móviles.

El modelo de la figura 8.2 es sólo un ejemplo, en muchos casos hasta puede que sea demasiado complejo y nos conformemos con utilizar únicamente la capa de Servlets/JSP para acceder a nuestros sistemas de información, en otros casos prescindiremos de la capa web, y en otros utilizaremos servicios web para acceder a la lógica de negocio, o crearemos capas de persistencia intermedia con patrones de acceso a datos. Las combinaciones son prácticamente ilimitadas.

Las espcificaciones y tecnologías Java EE animan la diversidad arquitectónica haciendo pocas suposiciones acerca de los detalles de las implementaciones de las APIs. Las decisiones y elecciones a nivel de aplicación son, en última instancia, un compromiso entre la complejidad y riqueza funcional.

Un escenario típico, donde aparecen todas las capas, es el escenario desde un navegador web. La generación de contenidos dinámicos se realiza normalmente en páginas JSP. La capa de EJBs nos

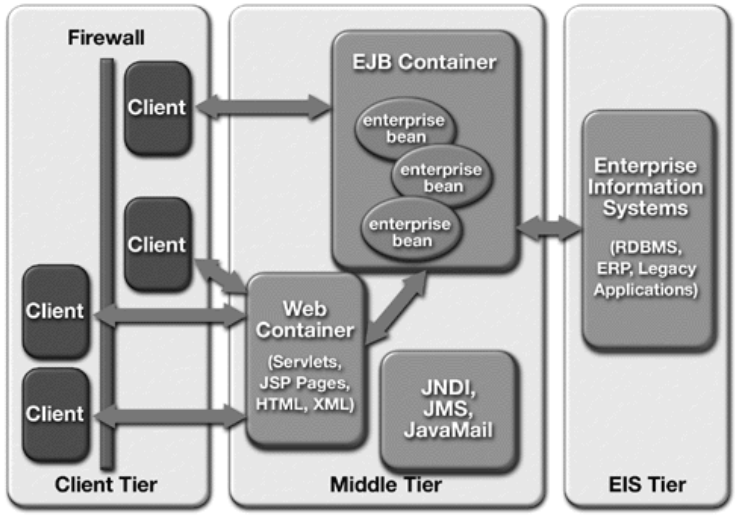


Figura 8.2: Esquema de modelo mixto de 3 y 4 capas

permite desacoplar el acceso a datos EIS de la interacción final con el usuario que se produce en las páginas HTML y JSP, como muestra la figura 8.3.

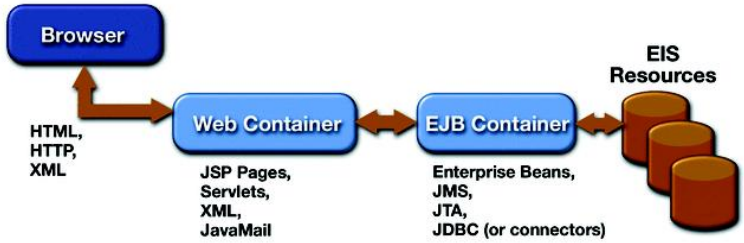


Figura 8.3: Escenario desde un navegador

La plataforma Java EE no obliga a usar en un sistema todas las capas. Lo esencial es escoger el mecanismo adecuado para cada problema. En este sentido, en ocasiones no existe la complejidad como para requerir una capa EJB como el de la figura 8.3. Se denomina **escenario centrado en la capa web** (figura 8.4) porque el contenedor web es el que realiza gran parte del trabajo del sistema. En una aplicación centrada en la capa Web sigue existiendo, aunque sea ligeramente, un modelo, que contiene entidades y reglas de negocio; es decir, el que no sean necesarios los EJB no implica no modularizar, mezclarlo todo y eliminar los componentes del modelo.

Aunque no entraremos a describir cada capa de una arquitectura Java EE en general, sí que nos vamos a detener en la capa web por su importancia en el presente proyecto. En esta capa el contenedor web proporciona servicios relacionados con las peticiones web, como muestra la figura 8.5. La plataforma Java EE se ejecuta encima de la plataforma J2SE, que a su vez viene soportada por el

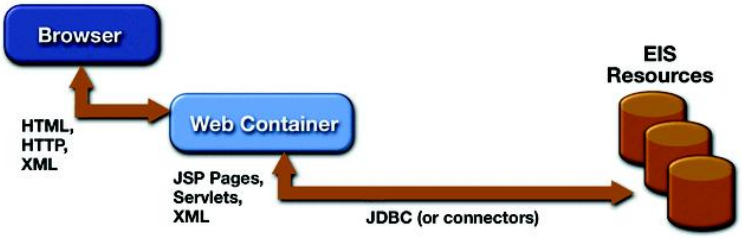


Figura 8.4: Escenario basado en web

sistema operativo. El código específico de la aplicación suele escribirse en términos de la capa de algún framework, que suministrará funcionalidades generales tales como despachar peticiones, invocar métodos del modelo o seleccionar y componer vistas, por ejemplo.

Figura 8.5: Estructura de la capa web

Dentro de las mejores prácticas recomendadas en las *BluePrints*, se nos dice que optemos por un framework existente (y probado), mejor que diseñar y construir el nuestro directamente. Entre las ventajas que supone este enfoque tenemos que: desacopla la presentación y su lógica, separa los roles de los desarrolladores, da un un punto central de control, facilita las pruebas unitarias y su mantenimiento, suele reducir los tiempos y costes, simplifica algunas tareas (como la internacionalización, validación, etc), puede ser compatible con otras herramientas lo que facilita su integración, etc.

8.3. Patrón MVC (Modelo-Vista-Controlador)

Además de proponer usar un framework existente, las *BluePrints* también recomiendan usar el patrón de diseño MVC (Modelo-Vista-Controlador) en aplicaciones interactivas. El seguimiento del paradigma MVC facilita la construcción y mantenimiento de aplicaciones web, y permitirá el uso de determinados componentes de una forma sencilla y flexible. El patrón MVC organiza la aplicación en tres módulos bien separados:

- El modelo de la aplicación con su representación del dominio y lógica de negocio,
- Las vistas que procuran la presentación de datos y la captura de datos por parte del usuario,
- Y el controlador que despacha las peticiones y mantiene el flujo del control.

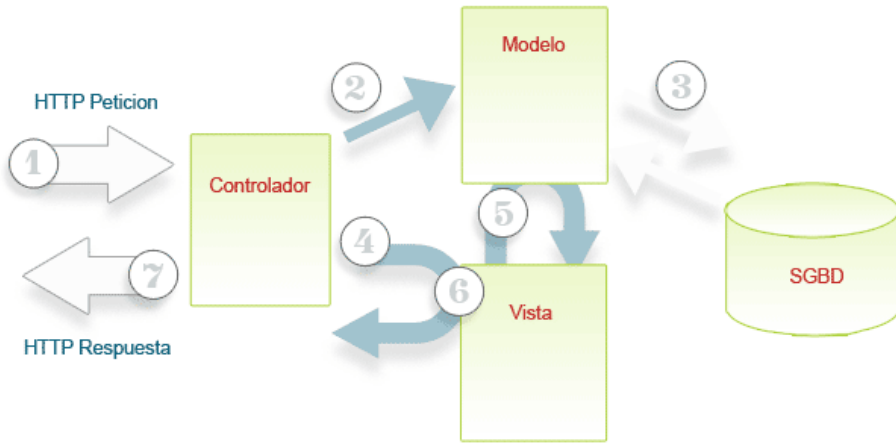


Figura 8.6: Patrón MVC: ciclo de servicio en la capa web

A partir de estos módulos el flujo que sigue a cualquier petición, ilustrado en la figura 8.6, suele ser el siguiente:

- El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)
- El controlador recibe (por parte de los objetos de la vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega.
- El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario).
- El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos (bien a través del modelo o dejando que el controlador envíe los datos

del modelo a la vista) para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra).

- La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

8.4. Arquitectura empleada en Virtual Museum

A lo largo de las sucesivas iteraciones del desarrollo del proyecto, la arquitectura se ha ido evolucionando en cuanto al conjunto de componentes usados y sus relaciones. Haber escogido la plataforma Java EE frente a otras nos ha traído diversas ventajas:

- Soporte de múltiples sistemas operativos: Al ser una plataforma basada en el lenguaje Java, es posible desarrollar arquitecturas basadas en Java EE utilizando cualquier sistema operativo donde se pueda ejecutar una máquina virtual Java.
- Organismo de control: La plataforma Java EE está contolada por el JCP, un organismo formado por más de 500 empresas. Entre las empresas que lo forman estan todas las más importantes del mundo informático (SUN, IBM, Oracle, SAP, HP, AOL, etc.) lo que garantiza la evolución de la misma.
- Madurez: Creada en el año 1997 como respuesta a la tecnología MTS de Microsoft, Java EE tiene ya más de diez años de vida y una gran cantidad de proyectos importantes a sus espaldas.
- Soluciones libres: En la plataforma J2EE es posible crear arquitecturas completas basadas única y exclusivamente en productos de software libre. No sólo eso, sino que los arquitectos normalmente disponen de varias soluciones libres para cada una de las partes de su arquitectura.

La arquitectura general, resultante de varios refinamientos y evoluciones, planteada para el desarrollo del proyecto se puede ver como un sistema en seis capas, reflejada en la figura 8.7, a saber:

- Capa de cliente: representa el interfaz de usuario que maneja el cliente.
- Capa de presentación web (vista y acciones): representa el conjunto de componentes que generan la información que se representará en el interfaz de usuario del cliente.
- Capa de servicios: provee los mecanismos necesarios para permitir la comunicación de Virtual Museum con otros sistemas externos mediante servicios web.
- Capa de lógica de negocio: contiene nuestros componentes de negocio reutilizables.
- Capa de acceso a datos o integración: aquí se encuentran componentes que nos permiten hacer más transparente el acceso a la capa de sistemas de información.
- Capa de sistemas de información: esta capa engloba a nuestros sistemas de información, tanto desde un punto de vista relacional (dada la naturaleza de la base de datos), como orientado a objetos.

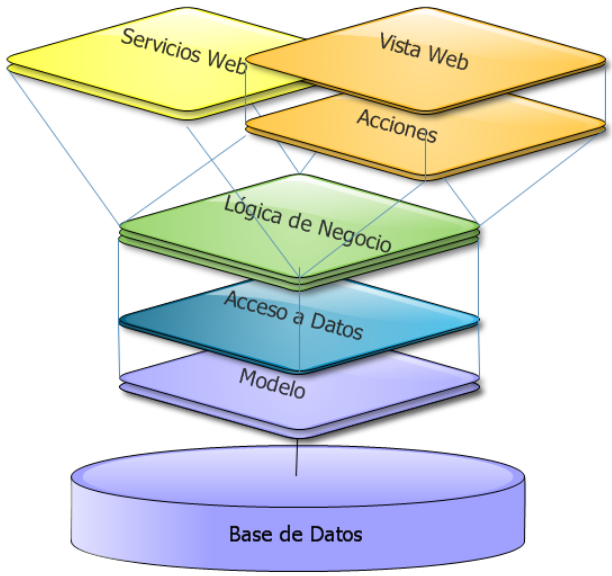


Figura 8.7: Arquitectura de capas en Virtual Museum

A continuación veremos más en detalle las funciones de cada una de las capas y las decisiones de diseño arquitectónico que se han adoptado.

8.5. Capa de Cliente

Nuestro sistema se va a componer de tres aplicaciones desde el punto de vista web, una por rol del sistema. Las aplicaciones para el responsable del museo y el administrador son web por lo que usarán un navegador y su interacción con el sistema empezará a partir de la capa de vista web.

El sistema desde el punto del visitante cuenta con tres interfaces bien diferenciadas:

- Público general: la forma habitual de acceso sería mediante navegador ya que el museo virtual generado crea un sitio web.
- Aplicaciones externas: se ofrece el acceso a recursos a aplicaciones de terceros mediante servicios web por lo que la comunicación con el sistema general comenzará por la capa de servicios web.
- Dispositivos móviles: el responsable del museo puede generar vistas personalizadas para el visitante en forma de aplicación Java *standalone* para dispositivos móviles.

8.6. Capa de Vista de presentación

8.6.1. HTML, CSS y Javascript

En las aplicaciones web que constituyen el sistema la vista se compone de HTML, CSS (hojas de estilo en cascada) y Javascript. Desde el punto de vista de HTML exclusivamente, hay que mencionar que durante la maquetación se ha tenido en cuenta que las páginas validen los estándares del W3C. De este modo, se ha intentado que el código HTML sea válido al menos según el esquema *XHTML 1.0 Transitional*.

En cuanto a la composición se ha puesto especial incapié en el uso de CSS, frente al de tablas (elementos *table* de html), que se dejan únicamente para la presentación de datos tabulares. En general el uso de tablas hace que las páginas sean más pesadas, no promueven la consistencia visual del sitio y no favorecen la accesibilidad. Por otra parte, las hojas de estilo presentan las siguientes ventajas:

- Accesibilidad: la separación de forma y contenido permite acceder a las personas con discapacidades a los contenidos de un sitio. Los navegadores permiten a los usuarios especificar su propia hoja de estilo local que será aplicada a un sitio web, por lo que según las necesidades del usuario, éste podría configurar su propia hoja de estilos para aumentar el tamaño del texto o remarcar más los enlaces.
- Menor cantidad de código, que redunda en menores tiempos de carga: jugando con el posicionamiento es posible presentar unas partes del contenido antes que otras dando aún mayor sensación de velocidad.
- Mantenimiento: el cambio de aspecto del sitio resulta mucho más sencillo al estar todas las normas de presentación ubicadas en un punto único.
- Futuro: La maquetación con tablas es cosa del pasado. Si todos los fabricantes se han dado cuenta de la importancia de los estándares y los adoptan, estamos garantizando una viabilidad a largo plazo de nuestros trabajos. Hay que ponerse sobre el camino adecuado.
- Independencia del dispositivo: Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o incluso a elección del usuario. Por ejemplo, para ser impresa, mostrada en un dispositivo móvil, o ser ”leída” por un sintetizador de voz.
- Gestión: el contenido se presenta agrupado basándose en criterios lógicos gracias a la utilización de etiquetas *div*, pudiendo presentarse un módulo de contenidos con diferentes aspectos según la página desde la que es llamado o incluso no presentarlo.

Por otra parte, con la intención de mantener nuestras páginas accesibles pero seguir usando Javascript se ha procurado hacer uso de Javascript no intrusivo. Este concepto se basa en conseguir que el código Javascript se abstraiga de su propia capa (comportamiento) superponiéndose al resto de capas sin estar mezclado con éstas. Al igual que los estilos deben ir separados del contenido, el comportamiento de los elementos de una página también debería estar separado de la estructura de la misma. Con ello conseguimos una ventaja ética y técnica importantisima en web: el extra es el JavaScript, y la web es completamente funcional aunque nuestro agente de usuario (habitualmente el navegador) no disponga de soporte Javascript o no lo tenga activado.

8.6.2. Internacionalización

La internaciónalización de mensajes se da a nivel de navegador en las aplicaciones del administrador y responsable del museo. Ambas aplicaciones tienen asociado un conjunto de mensajes por cada idioma que soportan. De esta forma, dado el idioma elegido, estas dos aplicaciones responderán mostrando los mensajes en dicho idioma. Si se quisiera que el sistema soportase otro idioma sólo habría que proponer el conjunto de mensajes que maneja el sistema traducidos al nuevo idioma en un fichero de recursos.

El museo virtual web, generado por la aplicación del visitante, por su parte no sigue el mismo camino de presentación de mensajes según el idioma elegido. En este caso, es el propio responsable del museo el que, según haya configurado su museo para soportar idiomas, escribirá el texto de cada contenido en los idiomas correspondientes. A partir de estos idiomas elegidos a voluntad, el museo virtual generado podrá habilitar un selector de idioma que devolverá los contenidos en el idioma elegido, pero siempre según lo redactado por el responsable del museo.

8.6.3. Struts Tiles

Habitualmente, un sitio web suele contener una conjunto de elementos de diseño comunes, tales como la cabecera, pie de página, menú, etc. En una aplicación web escrita en Java la vista típicamente se construye a partir de ficheros JSP, donde el planteamiento de un esquema de composición (*layout*) de la vista lleva a usar etiquetas *jsp:include* que proporcionan un mecanismo de reutilizar páginas y dar consistencia a nuestras aplicaciones. Aunque es una solución simple y se usa frecuentemente, en nuestro caso hemos preferido usar otra técnica, ya que ésta tiene un inconveniente, y es que si cambiamos el esquema de presentación de la aplicación tendremos que hacerlo para cada página lo que supone un cambio en la mayor parte de las ocasiones tedioso. Por tanto, aunque esta solución alcanza la reutilización de componentes, no lo logra con las plantillas y la lógica del esquema de presentación.

Siguiendo estas ideas, para las aplicaciones del administrador y responsable del museo hemos usando Struts Tiles. Mediante Struts Tiles podemos definir los esquemas como plantillas. Desde estas plantillas (*layouts*), podemos insertar marcadores de lugar en vez del actual componente de vista. Así, para todos los componentes esta página define un esquema reutilizable. La mayor ventaja de esta solución es que encapsula el comportamiento del *layout* y reduce drásticamente el acoplamiento entre los componentes.

Una poderosa característica de Tiles es la herencia entre definiciones. Por tanto, podemos crear una definición base y dejar que las demás hereden de ésta. La definición base solo debe suministrar la características comunes y la definición hija solo debe definir sus componentes propias.

8.6.4. Velocity

Por otra parte, para la vista de la aplicación del visitante podríamos haber seguido usando Struts Tiles o incluso plantear JSPs. Sin embargo, de nuevo esta aplicación presenta notables diferencias como para poder aplicar la misma solución. Para esta aplicación, uno de los requisitos es que el sitio

web generado ha de poder ser personalizable mediante plantillas diseñadas externamente y subidas al sistema, pero teniendo en cuenta que la plantilla no tiene porqué presentar esquemas para cada tipo de contenido. Para lograr esta funcionalidad, lo que buscamos en este caso es plantear un motor de plantillas que permita la composición de páginas de una forma dinámica y permitiendo el análisis de la plantilla dada para *renderizar* la vista incluida en ella o una por defecto. Además, de cara al diseñador de plantillas el sistema debe proporcionar un mecanismo de generación de textos en base a un conjunto de etiquetas predefinido. Si hubiésemos seguido usando Tiles alcanzar estos requisitos se habría vuelto más complejo. Al optar por el planteamiento de un motor de plantillas en base a otro ya existente se puede llegar a un esquema más flexible.

De entre los motores de plantillas libres disponibles en internet, la elección en nuestro caso se ha decantado por el uso de *Velocity*. Velocity es un motor de plantillas que permite a los diseñadores de páginas hacer referencia a métodos definidos dentro del código Java. Los diseñadores web pueden trabajar en paralelo con los programadores Java para desarrollar sitios de acuerdo al modelo MVC, permitiendo que los diseñadores se concentren únicamente en crear un sitio bien diseñado y que los programadores se encarguen solamente de escribir código de primera calidad. Velocity separa el código Java de las páginas Web, haciendo el sitio más mantenible a largo plazo. Para proveer una manera mas fácil, simple y limpia de incorporar contenido dinámico dentro de una página web se creó el VTL (Lenguaje de Plantillas de Velocity). VTL usa referencias para incluir contenido dinámico dentro de un sitio web. Una variable es un tipo de referencia que puede referirse a algo definido dentro del código Java mediante reflexión u obtener su valor de un enunciado VTL en la página misma.

Dentro de las diversas características de Velocity encontramos una particularmente útil para nuestros propósitos como son las macros. El elemento de script *#macro script* permite definir un segmento de plantilla VTL repetitivo. Las macros (también llamadas *velocimacros*) son muy útiles en una amplia gama se situaciones, tanto simples como complejas y nos brindará la flexibilidad que necesitamos para aportar una solución a nuestros requisitos específicos. De hecho, el sistema de componentes por defecto y conjunto de etiquetas (apéndice C) de generación de contenidos se desarrollará en base a macros.

8.7. Capa de Lógica de presentación

La capa de lógica de presentación nos va a brindar la conexión entre nuestra lógica de negocio y la presentación. Para ello, hemos optado por seguir un patrón MVC (Modelo-Vista-Controlador) como una elección natural para hacer esta separación. Según recomiendan las *BluePrints* vamos a hacer uso de un framework existente para esta capa. Como ya hemos visto en la capa anterior, los requisitos necesarios para las aplicaciones del administrador y responsable del museo por una parte, y la del visitante por otra presentan claras diferencias, por lo que la elección del framework empleado varía en un caso y en otro.

8.7.1. Framework Struts

En los últimos tiempos, el estándar de facto para el desarrollo de aplicaciones web ha sido Struts. El framework Struts proporciona a los desarrolladores una sólida base sobre la que construir sus aplicaciones web. Siguiendo la filosofía del patrón MVC, Struts separa completamente la lógica de

presentación de la lógica de negocio, lo que en este caso particular puede verse reflejado en la figura 8.8:

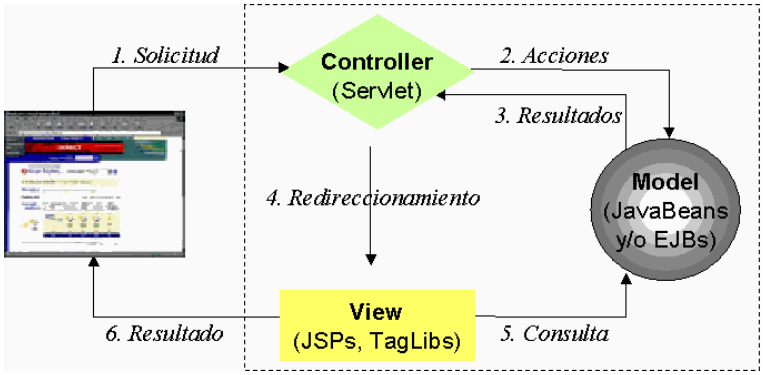


Figura 8.8: MVC en Struts

El navegador genera una solicitud que es atendida por el controlador (un *servlet* especializado). El mismo se encarga de analizar la solicitud, seguir la configuración que se le ha programado en su XML (típicamente en el archivo *struts-config.xml*) y llamar a la acción correspondiente pasándole los parámetros enviados. La acción instanciará y/o utilizará los objetos de negocio para concretar la tarea. Según el resultado que retorne la acción, el controlador derivará la generación de interfaz a una o más JSPs, las cuales podrán consultar los objetos del modelo (directa o indirectamente) a fines de realizar su tarea.

Struts implementa el servlet controlador dentro del patrón MVC pero nos deja total libertad tanto para el modelo como para la vista. De todas formas, si fuera necesario se puede heredar y ampliar o modificar. El flujo de la aplicación se puede programar desde un archivo XML. Las acciones que se ejecutarán sobre el modelo de objetos de negocio se implementan basándose en clases predefinidas por el framework y siguiendo el patrón Facade. Y la generación de interfaz se soporta mediante un conjunto de etiquetas (*tags*) predefinidos por Struts, además de las propias dadas por JSTL, cuyo objetivo es evitar el uso de *scriplets*[1], lo cual genera ventajas en cuanto a la mantenibilidad y eficiencia.

8.7.2. Framework Spring MVC

Para la aplicación del visitante, en vez de usar el framework Struts se ha seleccionado Spring MVC para esta capa. Aunque veremos más adelante una descripción de la pila de módulos de Spring en general, podemos adelantar aquí algunos aspectos del bloque MVC, que, como su propio nombre indica, implementa una arquitectura Modelo - Vista - Controlador.

Básicamente, el flujo de trabajo para atender peticiones se plasma en la figura 8.9. Como indica la figura, la petición (*request*) llega al *DispatcherServlet* el cual tiene la responsabilidad de delegar a otro componente (*Controller*) el procesamiento de la petición. Para obtener el nombre del controlador

[1] Porciones de código Java entre <% y %>, y que no suelen ser recomendados por los problemas de *casting*, baja reutilización y acoplamiento que presentan

que recibirá la petición, el *DispatcherServlet* le pregunta a uno o mas objetos *Handler Mapping* cuál será el controlador que recibirá la petición.

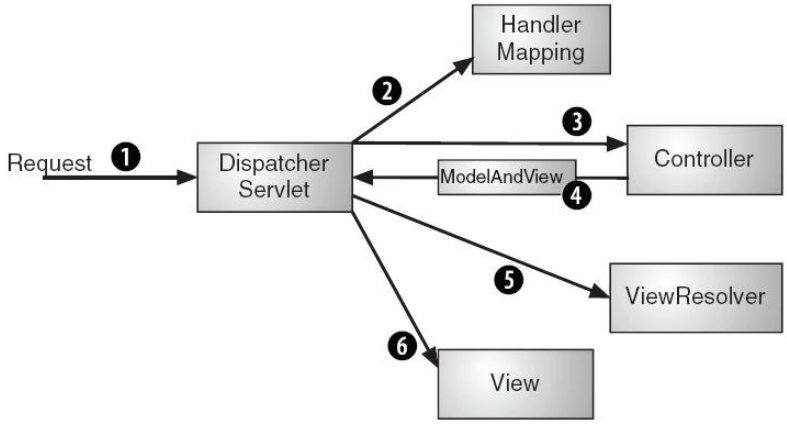


Figura 8.9: MVC en Spring

Para poder implementar un controlador sobre Spring es necesario que se cree una clase que herede de los Controller que han sido implementados por Spring, los cuales dependiendo de la funcionalidad a realizar será más adecuado elegir uno u otro.

Después de que el controlador reciba la petición, éste terminará construyendo un objeto *ModelAndView* que consiste en un nombre lógico de la vista que se va a *renderizar* y un modelo (que deriva de *Map*) conteniendo una serie de pares nombre de bean y su correspondiente objeto.

La vista devuelta por el controlador como parte del objeto *ModelAndView* es resuelta por el *ViewResolver*, que realiza la correspondencia entre este nombre lógico y el componente a utilizar, y la devuelve al usuario por medio del *DispatcherServlet*.

8.8. Capa de Lógica de negocio

Llegados a la capa de lógica de negocio, todo el sistema confluye a través de una implementación de clases única, centralizando así los componentes de lógica de forma general.

Frecuentemente, cuando se habla de sistemas Java EE, se piensa inmediatamente en *EJBs (Enterprise Java Beans)* como única forma de implementar la capa intermedia de lógica de negocios. Los *EJBs* son objetos reutilizables que contienen la lógica de negocio de nuestro sistema. Los contenedores de EJBs son servidores que se encargan de controlar todos estos componentes, esto es muy importante ya que se automatizan tareas como la gestión del ciclo de vida, la gestión de las transacciones, la gestión de persistencia, etc. Los desarrolladores, de este modo, no tienen que centrarse en crear servicios de bajo nivel y pueden centrarse en la creación de lógica de negocio empresarial.

Sin embargo, el desarrollo de un sistema Java EE no tiene porqué implicar el uso de EJBs, pudiendo adoptar otros enfoques como el propuesto para el presente proyecto: el framework Spring.

En primer lugar, hay que aclarar que Spring y EJB no son alternativas al mismo nivel estrictamente hablando: Spring es un framework de aplicaciones y EJB es una arquitectura de componentes para el desarrollo y despliegue de aplicaciones empresariales orientadas a objetos y distribuidas. Lo que hace que la comparación pueda resultar natural es que Spring fuera desarrollado como una reacción a EJB y que tengan algunas áreas de solapamiento.

Spring tiene algunas ventajas sobre EJB. Para empezar no necesita un servidor de aplicaciones, lo que permite trabajar directamente teniendo disponible un contenedor servlet o web; EJB también puede pero lo limitamos bastante en cuanto a servicios y otras áreas. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria. En general, esto redunda en una mayor productividad y mejor testabilidad, ya que al ser un contenedor ligero, el código de la aplicación queda más cercano a Java ”plano”, en vez de tener dependencias a APIs pesadas, y se pueden probar los diversos objetos fuera del propio contenedor.

Por otra parte, Spring es una gran herramienta de inyección de dependencias (es el motivo por el que nació), pudiéndose aplicar a todo tipo de objetos. Además, posee componentes como SpringAOP, que proporciona herramientas de programación orientada a aspectos más poderosas que los interceptores de EJB 3, o el Spring MVC que ayuda a integrar un patrón MVC en la aplicación.

Si bien el uso de un contenedor de EJBs 2.1 normalmente constituía una solución pesada, con un enfoque todo-nada, lo que no lo hacía adecuado para aplicaciones que no requiriesen todos los servicios que brindaba, la nueva especificación EJB 3.0 puede resultar adecuada en algunos casos, debido a la reducción de la complejidad de configuración, uso intensivo de anotaciones, persistencia a través del API JPA, etc. A pesar de ello, aunque con EJB 3 se mejoran diversos aspectos, el framework Spring puede resultar una solución potente y a la vez más liviana y flexible para el proyecto, por lo que se ha preferido su empleo en la construcción del sistema.

8.8.1. Framework Spring

Spring es un *framework* de código abierto, creado por Rod Johson, que permite desarrollar aplicaciones con funcionalidades similares a las basadas en EJBs, usando JavaBeans, o POJO’s (*plain old Java objects*), sin perder de vista la testabilidad o el mínimo acoplamiento. De forma resumida se puede decir que Spring es un contenedor ligero de inversión de control y orientado a aspectos. Esta breve definición resume las siguientes características:

- **Ligero:** Spring es ligero en terminos de tamaño y carga. De hecho, el framework completo de Spring se puede distribuir en un solo fichero JAR que pesa 1MB aproximadamente. Spring no es intrusivo: una aplicación bajo Spring no tiene, típicamente, dependencias con clases específicas de Spring.
- **Inversión de control:** Spring promueve un bajo acoplamiento a través de una técnica conocida como inversión de control (IoC). Cuando se aplica la inversión de control, a los objetos se les pasan pasivamente sus dependencias en vez de crear o buscar sus objetos dependientes ellos mismos. Así, el contenedor brinda las dependencias a cada objecto en tiempo de instanciación.
- **Orientado a aspectos:** Spring viene con un amplio soporte para la programación orientada a objetos que permite un desarrollo cohesivo separando la lógica de negocio de los servicios del

sistema (como la auditoría o la gestión de transacciones). Así, los objetos de la aplicaciones hacen lo que se suponen que deben hacer (la lógica de negocio para la que han sido diseñados) y nada más.

- **Contenedor:** Spring es un contenedor en el sentido de que contiene y gestiona el ciclo de vida y la configuración de los objetos de la aplicación. A pesar de ello, no debe confundirse con los contenedores de EJB's que suelen ser mucho más pesados en términos de tamaño y carga.

- **Framework:** Una aplicación Spring se puede ir configurando y construyendo a partir de componentes más simples. En Spring, los objetos se pueden componer de una forma declarativa, típicamente a través de un archivo XML. Además de esto Spring ofrece soporte para una variedad de subframeworks y conexión a otros frameworks que amplían la potencia de Spring.

La arquitectura de Spring se compone de siete módulos bien definidos. Aunque estos siete módulos en conjunto nos ofrece lo que necesitamos para desarrollar aplicaciones empresariales, no es necesario basar nuestra aplicación completamente en los siete módulos de Spring. Como se puede ver en la figura todos los módulos Spring se apoyan en el núcleo contenedor.

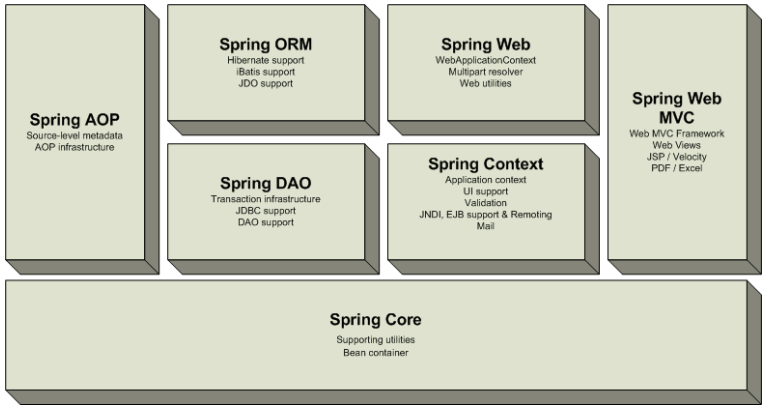


Figura 8.10: Framework Spring

- **Núcleo contenedor:** El contenedor es la parte esencial de Spring; define cómo se crean, configuran y gestionan los *beans*; implícitamente, se usan las clases de este módulo cuando se está configurando la aplicación, como por ejemplo *BeanFactory*, el corazón de un sistema Spring. Un *BeanFactory* es una implementación del patrón *factory* que aplica IoC para separar la configuración de la aplicación de las dependencias generadas entre los distintos bloques de código.

- **Contexto de Aplicación:** El *BeanFactory* del núcleo convierte a Spring en un contenedor, pero el módulo de Contexto es el que le hace ser un *framework*. Este módulo extiende el concepto de un *BeanFactory* añadiendo soporte para la internacionalización (i18n), eventos en el ciclo de vida de la aplicación, y validación. Además, ofrece servicios como email, JNDI, integración con EJB, remoting, etc. y en otro ámbito incluye soporte para la integración con *frameworks* de plantillas como FreeMarker o Velocity.

- **Programación orientada a Aspectos (AOP):** Spring ofrece un amplio soporte para la programación orientada a aspectos. Para asegurar la interoperabilidad entre Spring y otras plataformas AOP, este módulo se basa en la API definida por AOP Alliance (proyecto de código abierto que promueve la adopción de AOP y la compatibilidad entre las diversas plataformas AOP). Este módulo también introduce la programación de metadatos, mediante la cual se pueden añadir anotaciones a nuestro código fuente que indiquen a Spring cómo y dónde aplicar aspectos.

- **Abstracción JDBC y DAO:** El trabajo con JDBC suele resultar a menudo en un conjunto de líneas de código que se repiten a lo largo de toda la aplicación (abrir conexión, sentencia, resultado, cerrar conexión). Spring abstrae este conjunto de líneas repetitivo para que el código de acceso a base de datos sea más limpio y simple. Además, envuelve los errores de los SGBD en una capa de excepciones que procuran dar más significado que los propios SGBD. Por otra parte, este módulo usa el de AOP para proveer servicios de gestión de transacciones.

- ***Mapeador* Objeto/Relacional (ORM):** Este módulo de Spring no implementa su propia solución ORM, sino que brinda atajos para varios ORMs populares entre los se incluyen Hibernate, JDO o iBatis. La gestión de transacciones de Spring soporta cada una de estas soluciones ORM así como JDBC.

- **Contexto Web:** El módulo de contexto web se apoya en el módulo de Contexto de la aplicación, dando un contexto apropiado para las aplicaciones web. Además, soporta diversas tareas orientadas a web, como peticiones para transferencias de archivos (*multipart requests*), enlaces de los parámetros pasados por *Request* con objetos de negocio, y más específicamente integración con Struts.

- **Plataforma MVC:** Aunque se puede integrar fácilmente con otros *frameworks* MVC como Struts, Spring trae consigo su propia solución, que usa IoC para dar una separación entre la lógica del controlador y los objetos de negocio.

Inversión de control

De entre las características que posee Spring la inversión de control es particularmente importante dentro de este contenedor. Ya se ha mencionado la conveniencia de evitar que el código de la aplicación dependa del propio contenedor. Sin embargo, lograrlo no es una tarea trivial a priori: la mayor parte de los objetos tienen dependecias (objetos de negocio, acceso a datos, recursos, etc), que deben ser resueltas para su ejecución.

Para satisfacer las dependencias de los objetos sin introducir nuevas con el contenedor, Spring incorpora una solución denominada inversión de control. La inversión de control (IoC - *Inversion of Control*) es un concepto junto a unas técnicas de programación en las que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos o funciones.

Tradicionalmente el programador especifica la secuencia de decisiones y procedimientos que pueden darse durante el ciclo de vida de un programa mediante llamadas a funciones. En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir. Generalmente,

la inversión de control es un concepto importante en los frameworks y a veces se puede entender mejor a través del *principio de Hollywood*: ”No nos llames, ya te llamaremos nosotros a ti”.

El concepto de la IoC es un término bastante amplio y que puede ser implementado de diferentes formas; principalmente hay dos tipos:

- Búsqueda de dependencias: el contenedor provee métodos a los componentes y un contexto de búsqueda. Éste es el enfoque dado por EJB y Apache Avalon. Deja la responsabilidad a cada componente de usar la API del contenedor para buscar sus recursos y colaboraciones. Aquí la inversión de control está limitada al contenedor y los métodos que la aplicación puede invocar para obtener sus dependencias.
- Inyección de dependencias: los componentes no buscan, y en vez de ello, proveen métodos Java planos para permitir que el contenedor resuelva por ellos sus dependencias. En ese caso, es el contenedor el responsable de conectar los componentes, resolviendo los objetos y pasándolos a las propiedades del JavaBean o constructores. Cuando se usan las propiedades de un JavaBean se habla de *inyección de setter*, y cuando se emplean los argumentos del propio constructor, se habla de *inyección de constructor*.

En el caso de Spring la forma de inversión de control usada es la inyección de dependencias a través de *setters*. Este tipo de inversión presenta algunas ventajas al eliminar el código de búsqueda de la aplicación, evitar la dependencia con APIs del contenedor, y que no se necesita implementar interfaces especiales. Veamos un ejemplo:

```
public class ContenidoManagerImp implements ContenidoManager {
        private MuseoManager museoManager;
        public void setMuseoManager(MuseoManager museoManager)
        {
                this.museoManager = museoManager;
        }

        private ContenidoDAO contenidoDAO;
        public void setContenidoDAO(ContenidoDAO dao)
        {
                this.contenidoDAO = dao;
        }
        [...]
}
```

Los métodos *setter* son invocados inmediatamente después de que el objeto sea instanciado por el contenedor, antes de que trate cualquier método de negocio, evitando problemas de carrera relacionados con estas propiedades. De esta forma, el objeto *ContenidoManager* tendrá disponibles sus propiedades *museoManager* y *contenidoDAO* sin necesidad de escribir el código relativo a la búsqueda de estos objetos. Además, el aspecto que tiene esta porción de la clase es como un JavaBean ordinario, y no presenta llamadas específicas a APIs de Spring.

Desde el punto de vista de Spring, la configuración y conexión de los objetos del sistema suele expresarse en términos de XML, aunque también es posible implementar un bean propio que lea las definiciones de configuración. El formato puede considerarse bastante intuitivo, como muestra la siguiente porción del archivo de configuración:

```
<bean id="contenidoManagerTarget" class="com.museum4j.negocio.ContenidoManagerImp">
```

```
        <property name="museoManager"><ref local="museoManager"/></property>
        <property name="contenidoDAO"><ref local="contenidoDAO"/></property>
</bean>
```

En el ejemplo, nótese el uso del elemento *property* para fijar la propiedad correspondiente del JavaBean, y el elemento *ref* para resolver las dependencias a otros beans. Así, los *beans museoManager* y *contenidoDAO* estarán definidos de una forma similar en el mismo contenedor o incluso en otro relacionado.

Programación orientada a aspectos

Mientras que la inversión de control hace posible desacoplar componentes evitando dependencias, la programación orientada a aspectos permite capturar la funcionalidad que se usa a lo largo de la aplicación en los distintos componentes. La programación orientada a aspectos se suele definir como una técnica de programación que promueve la separación de intereses dentro de un sistema software. Los sistemas están compuestos por diversos componentes en los que cada uno se hace cargo de una parte específica de la funcionalidad global. A pesar de ello, a menudo estos componentes también conllevan alguna responsabilidad más allá de sus funciones básicas, como servicios de *logging*, gestión de transacciones o seguridad. Estos servicios del sistema presentes a lo largo de los distintos componentes introducen dos niveles de complejidad al código:

- El código que implementa estas funcionalidades está duplicado por múltiples componentes. Incluso si se abstrae la funcionalidad en un módulo separado para que el impacto sea una sóla llamada en el componente, dicha llamada sigue estando duplicada por múltiples puntos en el código.

- Los componentes se cubren con código que no se enmarca dentro del núcleo de su funcionalidad.

La figura 8.11 ilustra esta complejidad añadida: los objetos de la capa de lógica de negocio están envueltos íntimamente con el servicio de gestión de transacciones. No sólo cada objeto ”sabe” que sus transacciones están siendo gestionadas sino que cada objeto es responsable de llevar a cabo estas funciones por él mismo.

La programación orientada a aspectos permite modularizar estos servicios y aplicarlos a los componentes que deban afectar de forma declarativa. Esto produce una mayor cohesión entre componentes y que cada uno de ellos se centre en su propia funcionalidad, ignorando los servicios del sistema en los que está envuelto. Se puede pensar en los aspectos como si fueran capas que cubren los componentes de la aplicación, de forma que pueden ser aplicadas de una forma flexible sin que el núcleo de la aplicación sepa de su existencia.

Se puede ver un ejemplo de esta característica en el siguiente fragmento de código. En él se muestra cómo la aplicación tiene un método *generarMuseoMovil* que realizará diversas operaciones. Sería interesante que si, por la razón que sea, se produce una excepción toda la operación se rechazara globalmente para que que no se quede en un estado incoherente. Es decir, sería deseable que la operación se ejecutara atómicamente como si fuera una transacción. En Spring, indicar este aspecto se puede hacer de forma declarativa sin tener que incluir en el objeto de negocio código adicional.

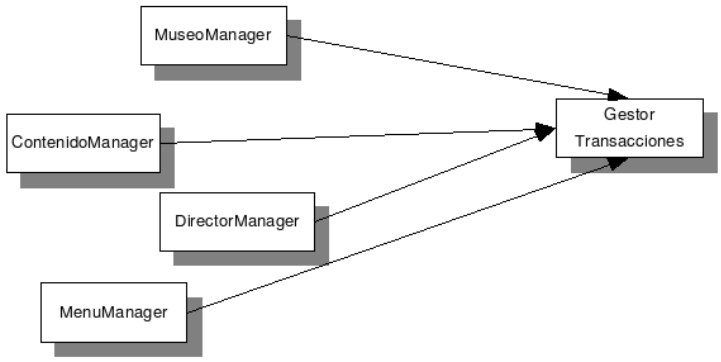


Figura 8.11: Gestión de transacciones

```
 <bean id="transactionManager"
     class="org.springframework.orm.hibernate3.HibernateTransactionManager">
     <property name="sessionFactory"><ref local="sessionFactory"/></property>
 </bean>

<bean id="contenidoManager"
     class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
     <property name="transactionManager"><ref local="transactionManager"/></property>
     <property name="target"><ref local="contenidoManagerTarget"/></property>
     <property name="transactionAttributes">
         <props>
             <prop key="guardar*">PROPAGATION_REQUIRED</prop>
             <prop key="actualizar*">PROPAGATION_REQUIRED</prop>
             <prop key="borrar*">PROPAGATION_REQUIRED</prop>
             <prop key="visitar*">PROPAGATION_REQUIRED</prop>
             <prop key="reordenar*">PROPAGATION_REQUIRED</prop>
             <prop key="quitarParada*">PROPAGATION_REQUIRED</prop>
             <prop key="setContenidoDePortada">PROPAGATION_REQUIRED</prop>
             <prop key="generarMuseoMovil">PROPAGATION_REQUIRED</prop>
             <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
         </props>
     </property>
 </bean>
```

En el ejemplo se hace uso del objeto de Spring *TransactionProxyFactoryBean*, que se trata de un *proxy* que permite interceptar llamadas a métodos de una clase y aplicar un contexto de transacciones. Además, en este caso se está haciendo uso de la clase *HibernateTransactionManager*; una implementación del gestor de transacciones frecuente si la capa de persistencia se basa en Hibernate.

8.9. Capa de Servicios

Además de las funcionalidades que va a presentar el sistema, se hace necesario publicar una serie de servicios accesibles por otras aplicaciones independientemente del lenguaje que se usó para

desarrollarlas. Es por eso por lo que el sistema expondrá un servicio web con una serie de operaciones para la interacción con otras aplicaciones.

En sistemas Java EE existen numerosos *frameworks* para el desarrollo de servicios web. Entre ellos se pueden citar: Axis1 y 2, XFire, Celtix, ActiveSOAP, GlassFish, Glue, XINS, etc. Aunque no existe una solución óptima, hemos optado por usar XFire. El *framework* XFire es una librería SOAP ligera de alto rendimiento que se basa en la filosofía de POJO's y no se centra en XML como tienden otras (es por eso por lo que XFire se anuncia como una plataforma para el desarrollo de servicios de próxima generación). En el desarrollo con XFire lo que hay que hacer es simplemente escribir los POJO's que van a actuar como servicios, marcar anotaciones según la especificación JSR-181, y configurar los servicios que se van a exponer en un archivo adjunto (*services.xml*). XFire hace el resto: introspecciona la interfaz del servicio y genera el WSDL a partir de él.

Además de las propias ventajas de XFire por sí sólo, también hay que hacer notar que se puede integrar con Spring fácilmente. De esta forma, podemos hacer uso de las características de inyección de dependencia en los propios servicios, para, por ejemplo, hacer que un servicio se apoye en un *bean* de la capa de lógica de negocio.

8.10. Capa de integración

Nuestra capa de lógica de negocio se basa en el manejo de entidades del modelo como objetos. La orientación a objetos en esta capa nos da mayor flexibilidad y comodidad a la hora de trabajar con nuestro modelo. Sin embargo, de cara a la capa de sistemas de información, las bases de de datos relacionales siguen siendo más utilizadas que las objetuales, por lo que necesitamos una capa de integración que lleve a cabo la transformación objeto-relacional o motor de persistencia (ORM - *object relation mapping*). Un motor de persistencia por tanto será una capa software cuya misión se centra en realizar de manera automática y transparente la traducción en ambos sentidos entre los objetos que modelan la lógica de negocio de la aplicación y las bases de datos relacionales donde se almacena aquella información que se desee perdurar.

8.10.1. Hibernate

Entre los motores de persistencia de código abierto podemos destacar Hibernate, Castor, Torque, OJB y Cayenne. De ellos, hemos elegido Hibernate como herramienta ORM ya que actualmente mantiene una reputación excelente en la comunidad de desarrollo gracias a sus prestaciones, buena documentación y estabilidad. Una característica de su filosofía de diseño es que no realiza una intrusión en la manera en la que se definen las clases de manera que tendrán el mismo aspecto que los utilizados en las aplicaciones normales. Hibernate utiliza un mecanismo proporcionado por Java que es la reflexión; gracias a ella es capaz de descubrir información sobre los atributos, métodos y constructores de las clases. De esta forma, Hibernate puede trabajar con los objetos comunes de Java, o también llamados POJOs (*Plain Old Java Objects*). Hibernate se apoyará en múltiples APIs conocidas en la actualidad. De esta forma, usará el API de JDBC internamente para conectarse a los distintos tipos de servidores de bases de datos datos relacionales.

Hibernate es una solución completa en sí misma, y aunque puede funcionar dentro de un conte-

nedor Web, no depende de él para llevar a cabo su labor persistente, ya que de hecho se puede utilizar en aplicaciones *standalone*.

Para indicar a Hibernate cómo se deben almacenar nuestros objetos en la base de datos se emplean archivos de *mapeo* XML. En un archivo de *mapeo* se especifican las propiedades del objeto ya sean propiedades del objeto o asociaciones con otros objetos. Podemos ver un ejemplo de cómo se configuraría una clase:

```
<class name="com.museum4j.modelo.Contenido" table="contenidos">
        <id name="idContenido" column="idContenido" type="java.lang.Integer">
                <generator class="increment" />
        </id>
        <many-to-one name="museo" column="museo" />
        <property name="fechaCreacion" column="fechaCreacion" type="java.util.Date"/>
        <set name="textos" table="textos" cascade="all" inverse="true" lazy="false">
                <key column="idContenido" />
                <one-to-many class="com.museum4j.modelo.Texto" />
        </set>
        <property name="hits" column="nVisitas" type="java.lang.Integer"/>
        [...]
</class>
```

Aunque la declaración de propiedades y asociaciones de cada objeto del modelo en ocasiones puede no ser tarea fácil, en el ejemplo anterior vemos cómo se pueden modelar propiedades simples (*número de visitas*), el propio identificador de la clase o clave primaria (*idContenido*), e incluso se pueden modelar relaciones N:1 (un contenido está asociado con un museo), y relaciones 1:N (un contenido presenta un conjunto de textos), entre otras. En este ejemplo, se muestra una parte de una declaración de *mapeos* de propiedades y asociaciones para una clase a modo de introducción, aunque en Hibernate (y así se ha necesitado para el presente proyecto) se pueden modelar otras relaciones más elaboradas como herencia, asociaciones ternarias, colecciones ordenadas, etc.

Aunque la capa de datos se implementa a través de Hibernate, su acceso desde la capa de lógica de negocio se hace a través de interfaces, desacoplando así la lógica del código que implementa la persistencia. Así, en cualquier momento se podría cambiar Hibernate por otro componente que maneje la persistencia de nuestro modelo hacia la base de datos.

8.11. Capa de sistemas de información

La última capa que da soporte a nuestro sistema es la que nos da la persistencia de la información que maneja. En este caso, los gestores de bases de datos relacionales se han consolidado de momento como las dominadoras del mercado, por lo que optamos por utilizar un gestor de este tipo. Entre los gestores de bases de datos relacionales *open source* existentes nos vamos a decantar por MySQL, frente a otros como Firebird, PostgreSQL, MaxDB o Ingres. Fundamentalmente, se usa MySQL debido a las siguientes características que presenta:

- Portabilidad: MySQL está soportado en la mayoría de los distintos "sabores"de Unix/Linux, así como Windows y MacOS X.
- Velocidad: Usa técnicas como mecanismos eficientes de indexado, tablas temporales en memoria y algoritmos optimizados para operaciones *join*, lo que le da una eficiencia y velocidad superior a otros sistemas de base de datos.

- Escalabilidad: Debido a su modularidad y su flexibilidad en su configuración, se puede ejecutar en un amplio abanico de máquinas desde ambientes embebidos hasta sistemas multiprocesador. Esta escalabilidad permite desarrollar nuestros sistemas sobre una configuración de desarrollo y luego portar la misma base de datos a otra máquina en producción. Además, al ser multihilo, MySQL aprovecha los recursos eficientemente para dar servicio concurrente a múltiples usuarios.
- Flexibilidad: Se permite elegir el tipo necesario de nuestras tablas para ajustarse a nuestros requisitos software.
- Facilidad de uso: Es relativamente fácil de instalar y administrar, comparado con otros sistemas de gestión de bases de datos.
- Seguridad: Se pueden restringir los permisos de acceso a los distintos usuarios pudiendo establecer derechos a distintos nivels: desde la base de datos completa hasta columnas.
- Acceso desde otros lenguajes/sistemas: Existen numerosas librerías y *APIs* para conectar MySQL con Java, C/C++, Perl, PHP, ODBC, etc.
- Licencia: Aunque en algunos casos se debe obtener una licencia comercial, MySQL se distribuye bajo licencia GPL, lo que en la mayoría de los casos permite usarse sin coste alguno.

8.12. Diagrama de Componentes

El patrón de diseño fundamental del sistema es MVC por lo que Struts representa el componente fundamental, dándonos el pilar principal sobre el que que conectar el modelo de datos con la vista. Sin embargo la inyección de dependencia que nos da Spring resulta muy atractiva por lo que se ha optado por hacer que coexistan los dos, a través del proxy *DelegatingActionProxy*. Aunque Spring también nos da un *framework* MVC, Struts sigue resultando mejor aproximación para las aplicaciones del administrador y responsable del museo.

Las ventajas de integrar un sistema Struts en la plataforma Spring son diversas. La primera de todas es que Spring se diseñó de forma explícita para resolver algunos de los problemas de la ”vida real JEE”, como la complejidad, baja eficiencia, testabilidad, etc.

Por otra parte, para la aplicación del visitante esta solución no resultaba totalmente apropiada por lo que en este caso, pasamos a usar otra combinación de componentes. En esta aplicación nos desligamos de Struts ya que desde el punto de vista de acciones, en este caso vamos a contar con un conjunto más reducido de acciones y Spring MVC nos ofrece mayor flexibilidad para nuestros objetivos. Sin embargo, sí que seguimos interesados en acceder a los beans de la lógica de negocio y que éstos accedan los objetos del modelo de la misma forma en que lo hacían las otras dos apliaciones. Por eso, aunque esta aplicación no comparte Struts como despachador de peticiones, y se deja únicamente en manos de Spring, sí que se siguen compartiendo los componentes de las capas inferiores.

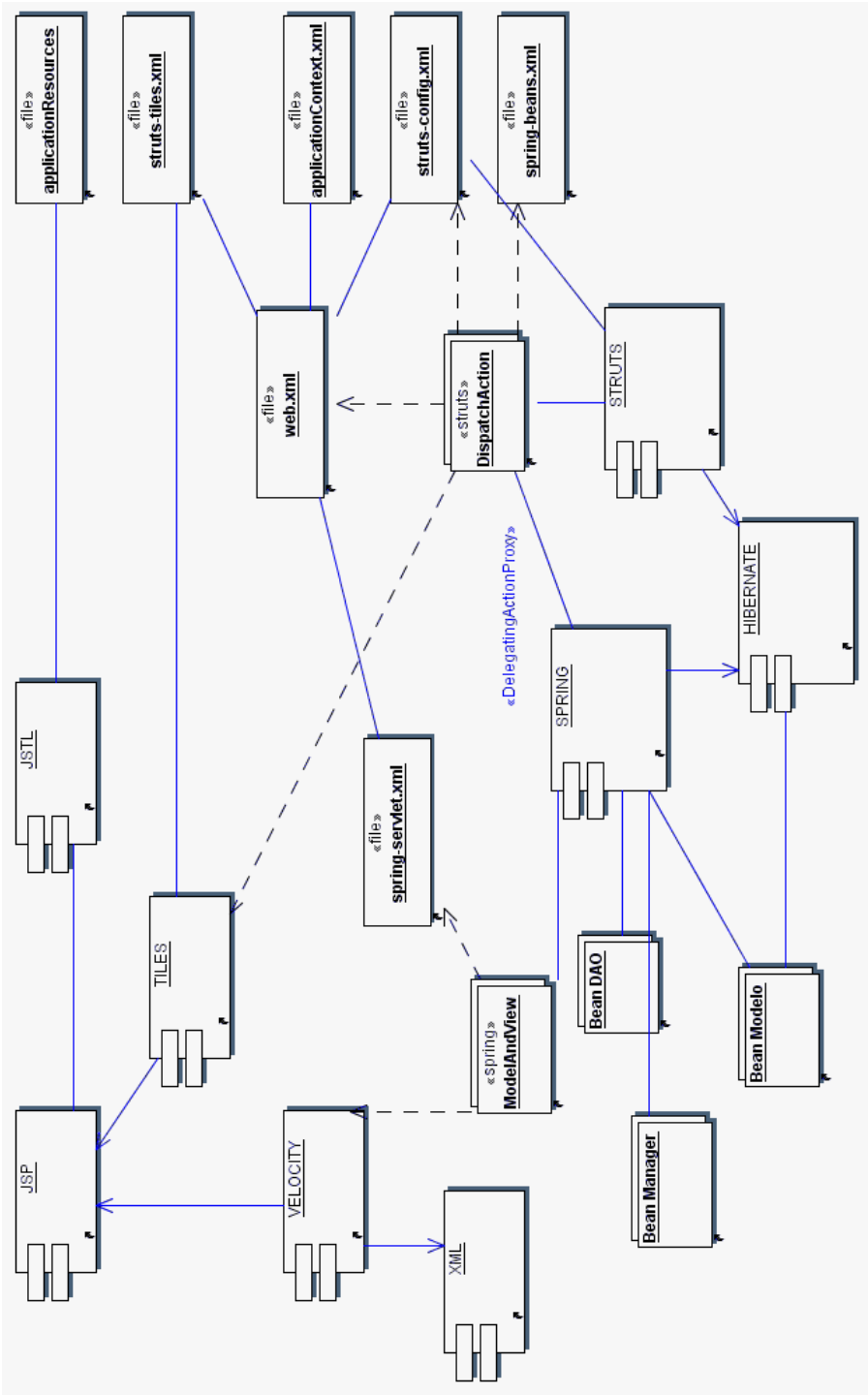


Figura 8.12: Diagrama de Componentes del sistema

Capítulo 9

DISEÑO

Los grandes conocimientos engendran las grandes dudas.

Aristóteles

Índice del Capítulo

9.1. Introducción

El modelo de clases de diseño describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además, el modelo de diseño sirve como abstracción de la implementación del sistema y es, de ese modo, utilizada como entrada fundamental de las actividades de implementación.

9.2. Diagrama de Clases de Diseño

A continuación se presentan los diagramas de clases de diseño organizados en paquetes según las capas que forman la arquitectura de la aplicación.

9.2.1. Diagrama de paquetes

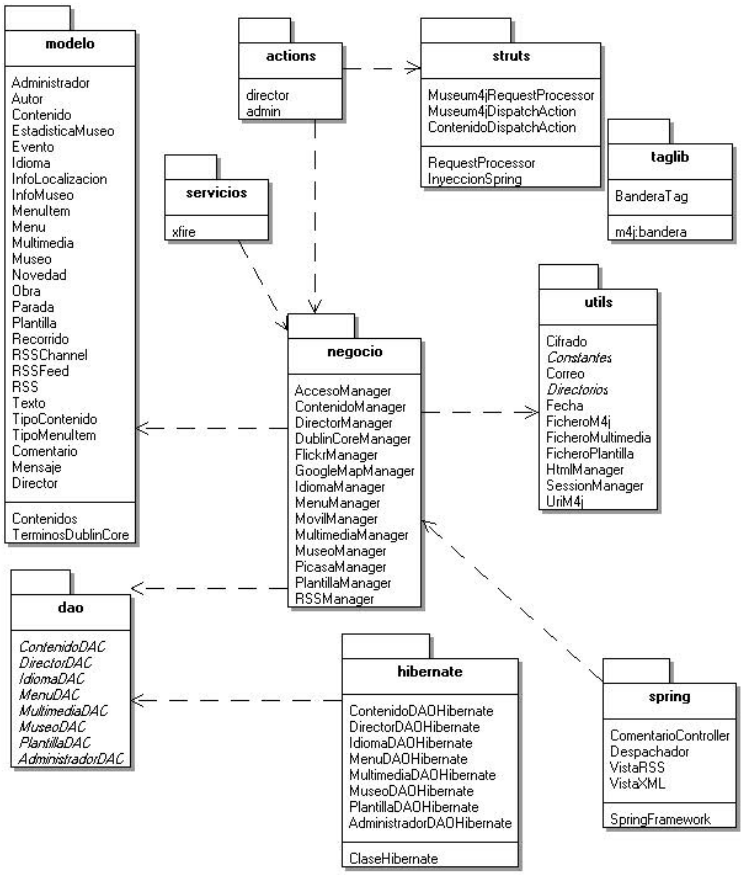
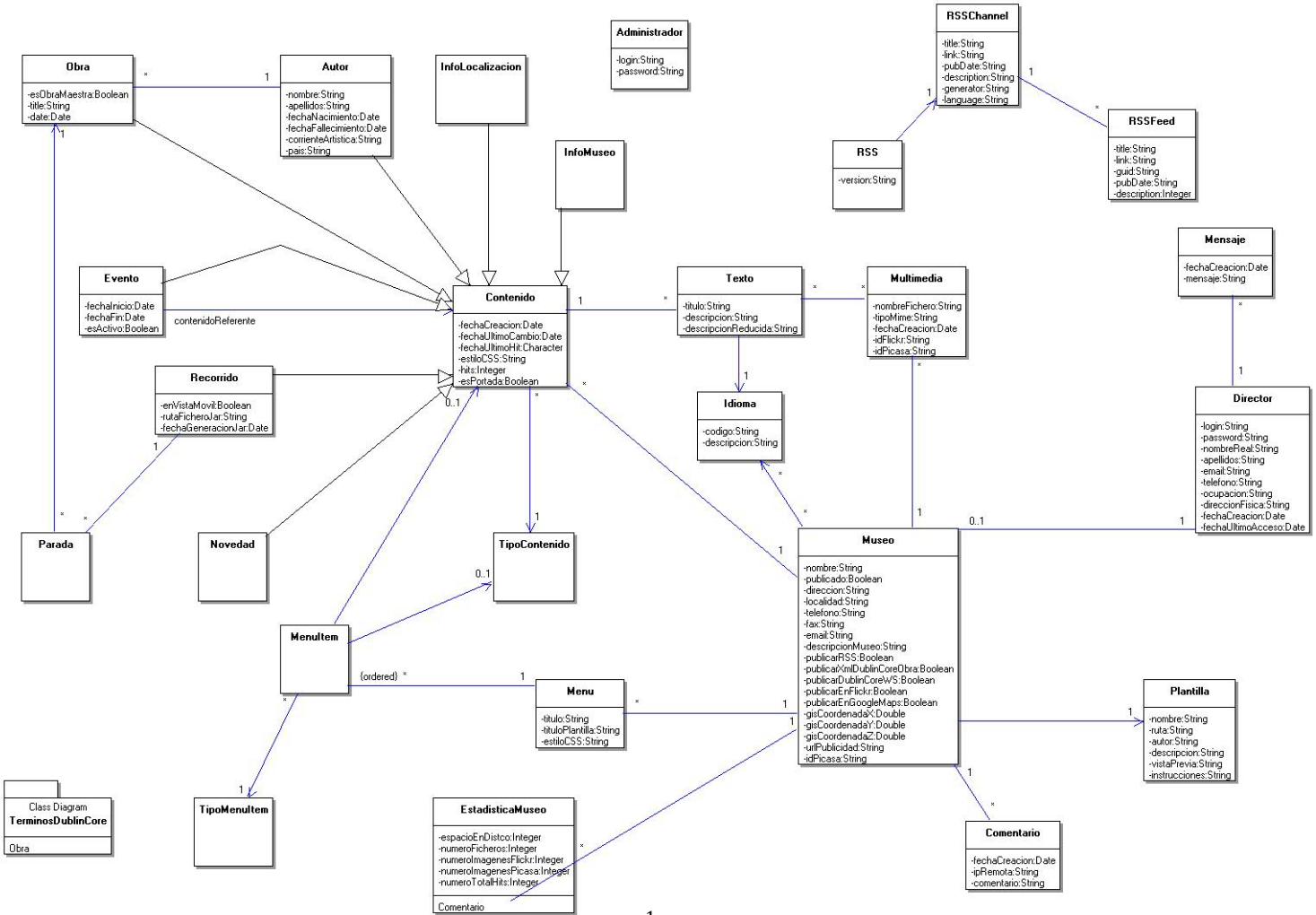


Figura 9.1: Diagrama de Paquetes de Diseño

9.2.2. Modelo

Las clases del paquete del modelo por sus dimensiones se presentan en la siguiente página.

Figura 2: Diagrama de Clases de Diseño - Paquete Modelo



9.2.3. Acceso a datos

La capa de acceso a datos está compuesta por una capa de interfaces (paquete DAO) y la implementación de las mismas usando el ORM Hibernate (paquete Hibernate).

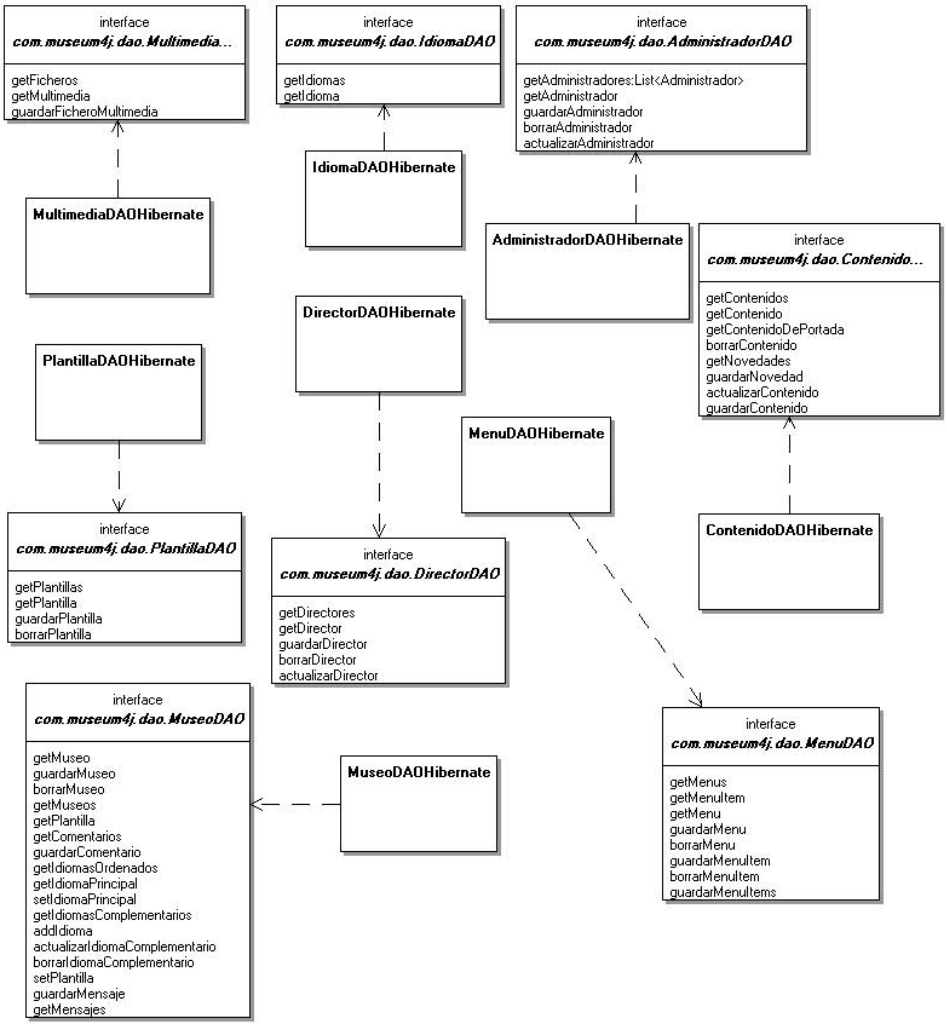


Figura 9.3: Diagrama de Clases de Diseño - Paquete Hibernate y DAO

9.2.4. Negocio

La capa de lógica de negocio se ha diseñado planteando, como en la capa de acceso a datos, como la composición de dos capas, partiendo de una primera capa de interfaces de forma que se desacopla la especificación de la clase de su implementación:

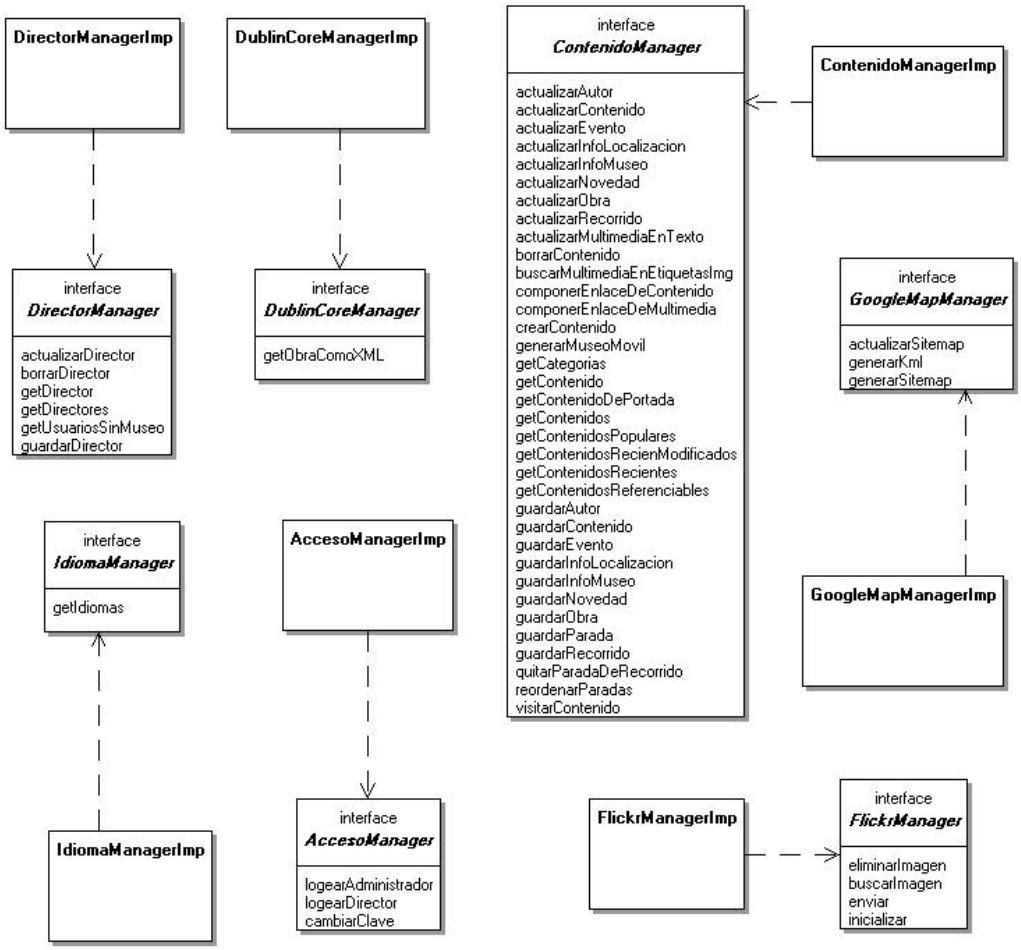


Figura 9.4: Diagrama de Clases de Diseño - Paquete Negocio I

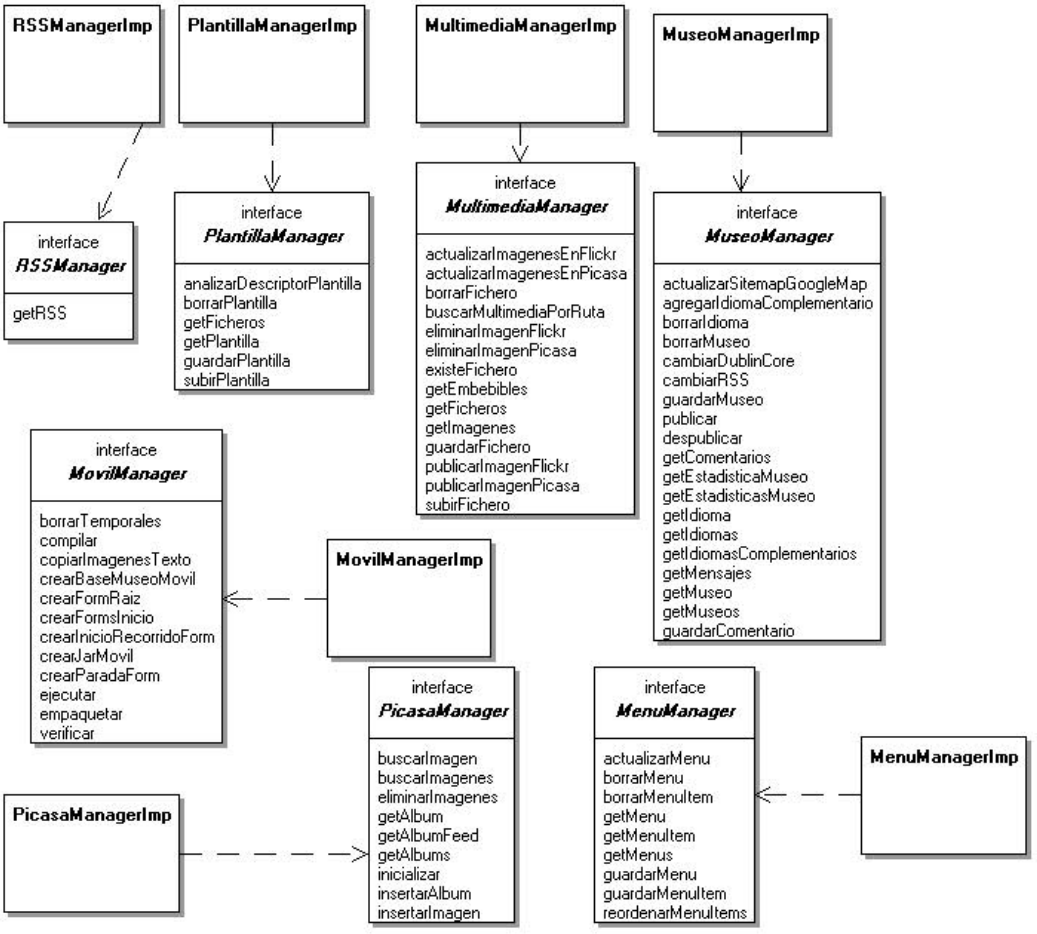


Figura 9.5: Diagrama de Clases de Diseño - Paquete Negocio II

9.2.5. Servicios Web

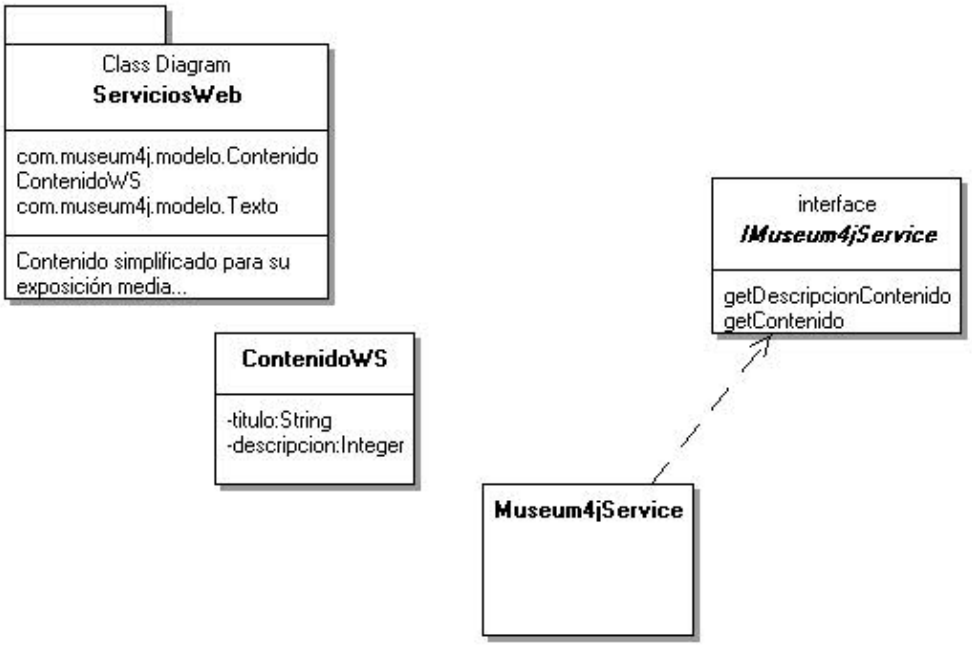


Figura 9.6: Diagrama de Clases de Diseño - Paquete Servicios Web

9.2.6. Acciones Visitante

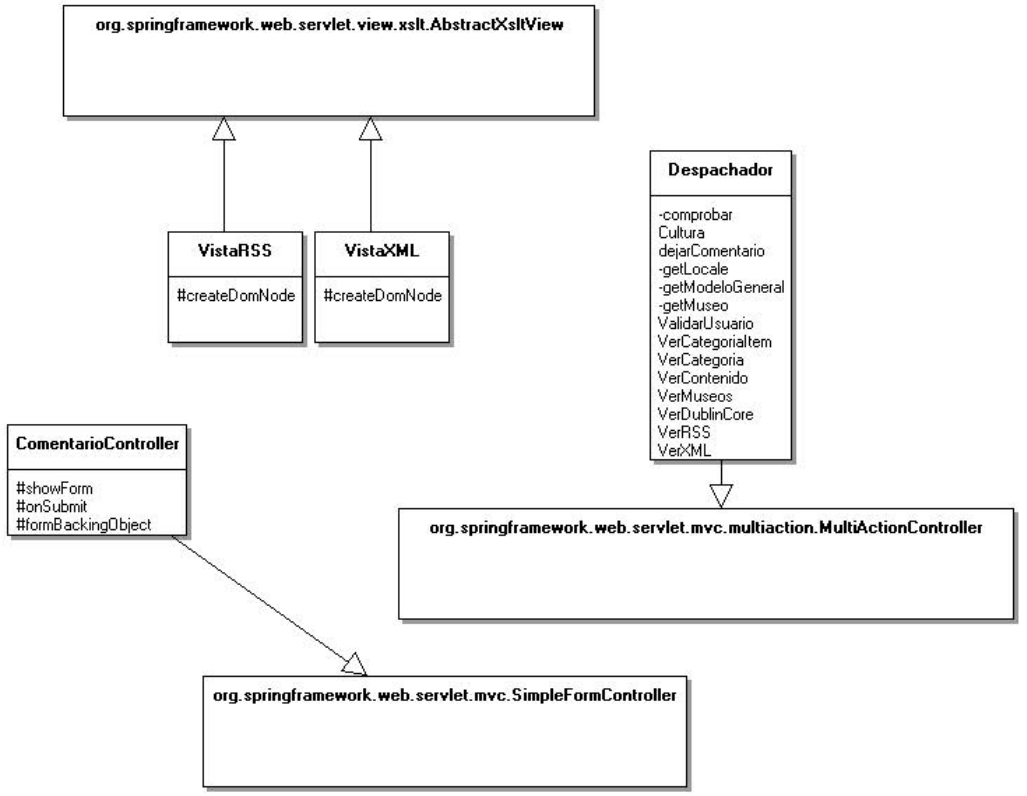


Figura 9.7: Diagrama de Clases de Diseño - Paquete Acciones Spring Visitante

9.2.7. Acciones Director

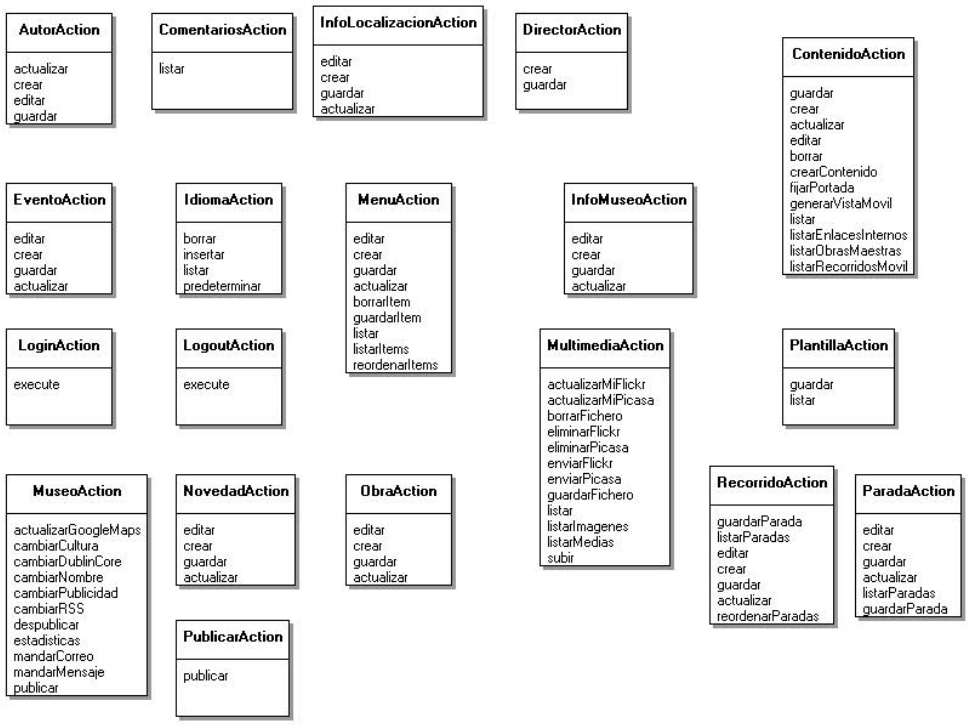


Figura 9.8: Diagrama de Clases de Diseño - Paquete Acciones Struts Director

9.2.8. Acciones Administrador

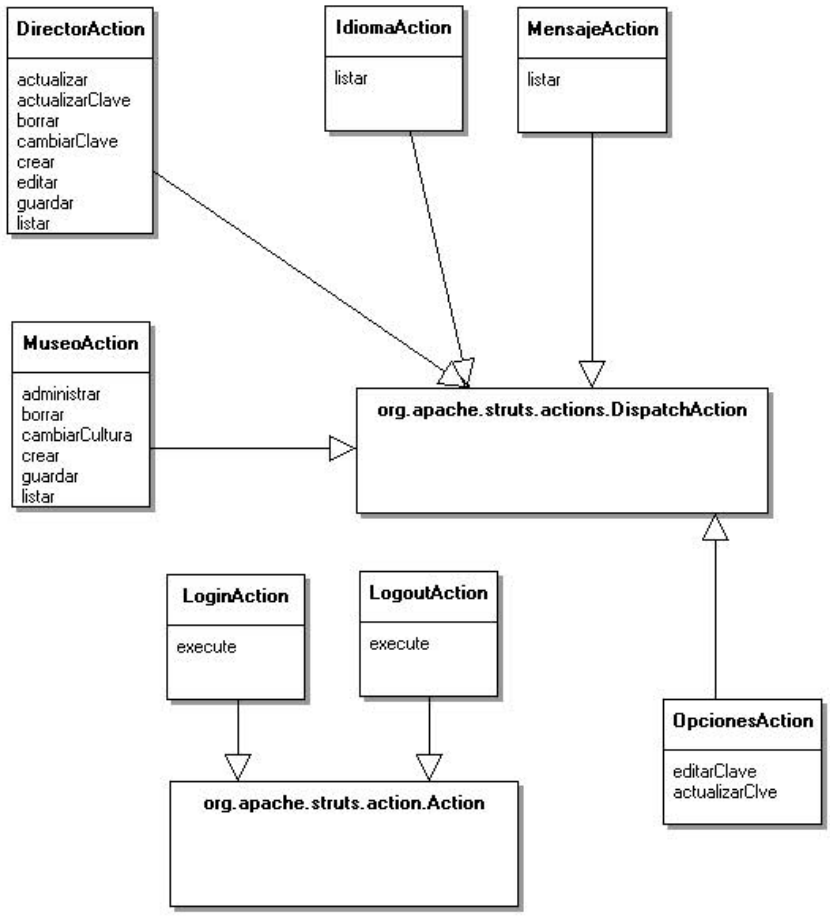


Figura 9.9: Diagrama de Clases de Diseño - Paquete Acciones Struts Administrador

9.2.9. Vista Móvil

Conjunto de clases y plantillas para la generación de la vista móvil.

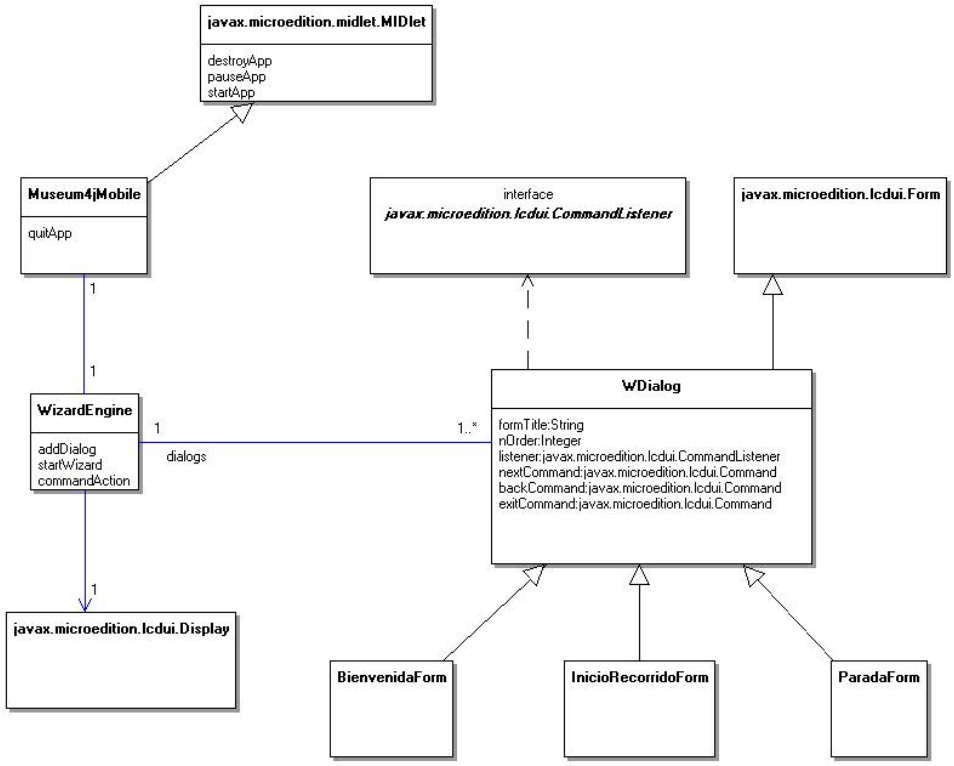


Figura 9.10: Diagrama de Clases de Diseño - Paquete VistaMovil

9.2.10. Taglib Bandera

Usado para los jsp de contenidos

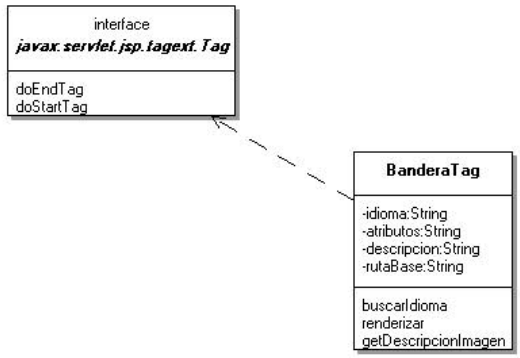


Figura 9.11: Diagrama de Clases de Diseño - Paquete Taglib

9.3. Realización de casos de uso-diseño

Las realizaciones de casos de uso-diseño se presentan en forma de diagramas de interacción, y más concretamente como diagramas de secuencia. Para su visualización puede dirigirse al CD anexo y abrir el modelo de diseño dentro del directorio ”together”.

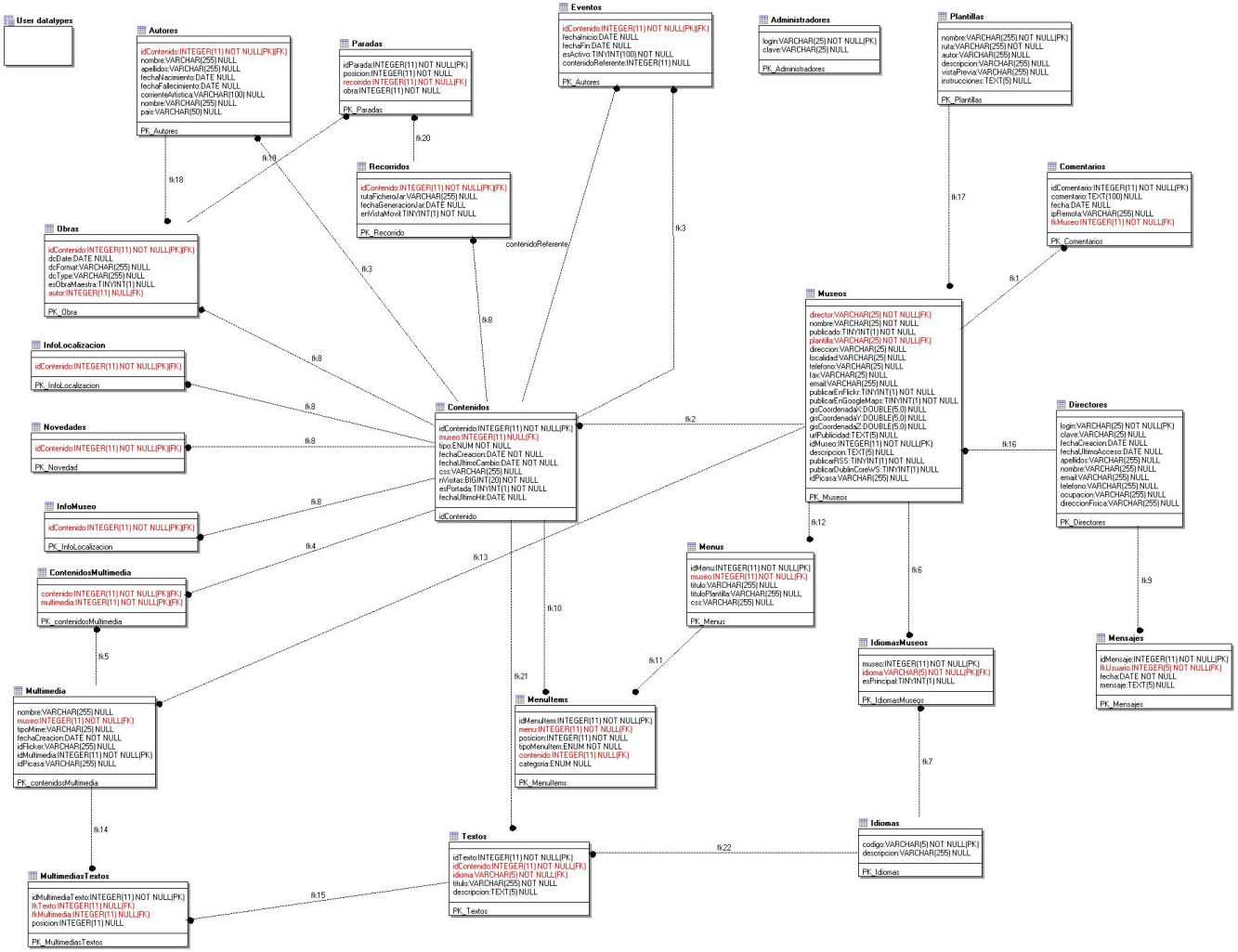
9.4. Descripción de operaciones

Hay ciertas operaciones o métodos de las clases de diseño que se han descrito durante la fase de diseño, mientras que el resto se han dejado como labor del propio Ingeniero de Componentes pero dentro de la fase implementación.

9.5. Diagrama Entidad-Relación

El diagrama entidad-relación que se usará para persistir las clases del paquete del modelo se presenta en la siguiente página, dadas sus dimensiones.

Figura 12: Diagrama Entidad-Relación general



9.5.1. Elementos Dublin core

Véase la figura 9.13.

9.5.2. Diccionario de datos

El diccionario de datos se mantiene en la propia documentación de las clases del paquete del modelo. Además, como se incluyen los archivos de *mapeo* de Hibernate se puede obtener una correspondencia directa entre los atributos de las clases y las columnas de las tablas correspondientes.

9.5.3. Modelo relacional

El modelo relacional puede obtenerse directamente de los propios scripts SQL de creación de la base de datos, y puede consultarse en el CD anexo a la memoria.

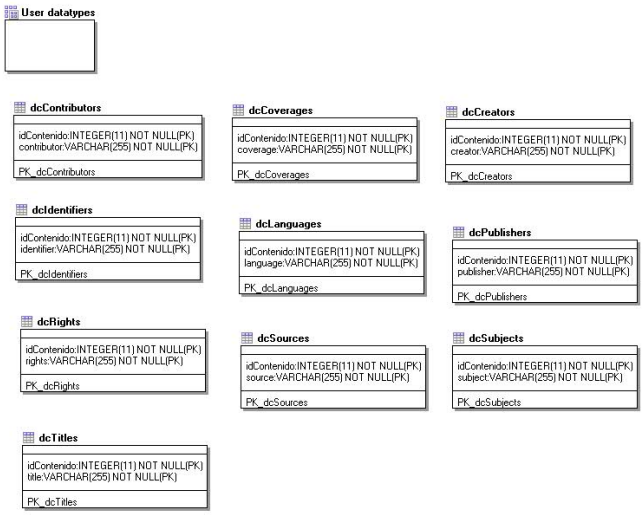


Figura 9.13: Diagrama Entidad-Relación de los Términos Dublin Core asociados a una obra

9.6. Diseño de Temas/Plantillas

Las plantillas que usan los museos virtuales siguen una estructura fija:

- Descriptor de la plantilla en XML.
- Imagen con la vista previa.
- Archivo contenido.vm que define la plantilla cuando se muestran contenidos concretos.
- Archivo categoria.vm que defina la plantilla cuando se muestran una categoría de contenidos.
- Otros archivos que incluye la plantilla tales como hojas de estilo CSS, imágenes u otros elementos usados bien por los archivos .vm o por las hojas de estilo.

9.6.1. Descriptor XML de la Plantilla

Las plantillas que maneja el sistema Virtual Museum tienen un conjunto de atributos determinado que se corresponden uno a uno con los nodos que debe presentar el archivo descriptor para su correcta importación. De ahí que los atributos que tiene la clase Plantilla:

- nombre
- autor
- descripción
- instrucciones

se reflejen fielmente en el descriptor XML de la misma:

```
<museum4j>
        <nombre>Mi primera plantilla</nombre>
        <autor>Francisco</autor>
        <vistaPrevia>miniatura.jpg</vistaPrevia>
        <descripcion>
                Esta plantilla es un ejemplo
        </descripcion>
        <instrucciones>
                Instrucciones de estilo adicionales para el correcto uso de esta plantilla
        </instrucciones>
</museum4j>
```

9.7. Macros de plantilla

Para aprovechar las posibilidades del motor de plantillas que brinda Virtual Museum al diseñador gráfico se ha ideado ofrecer una serie de palabras clave que dentro de un museo virtual generado se transforman en distintos textos dependiendo de la macro en particular. La referencia de las plantillas que pueden ser usadas se describen en el apéndice C.

Capítulo 10

IMPLEMENTACIÓN

Un experto es una persona que ha cometido todos los errores posibles en un campo muy acotado y, por tanto, es el que tiene experiencia, ha acumulado casos y es capaz luego de rescatarlos en el momento que se enfrenta a situaciones parecidas.

Niels Bohr

Índice del Capítulo

10.1. Introducción

En la implementación empezamos con el resultado del diseño e implementamos el sistema en términos de componentes de código fuente y recursos asociados. Llegados este punto, la mayor parte de la arquitectura del sistema es capturada durante el diseño, siendo el propósito principal de la implementación el desarrollar la arquitectura y el sistema como un todo.

Aunque el artefacto que produce esta fase es básicamente el conjunto de archivos de código fuente y recursos asociados para su construcción, despliegue y ejecución, en este capítulo de la memoria, hemos querido hacer notar ciertos aspectos presentes en el código desarrollado que merecen cierto interés.

10.2. Componentes y Algoritmos

10.2.1. Búsqueda de multimedia en textos

La facilidad que se le ofrece al responsable del museo para editar los textos asociados a sus contenidos viene dada gracias a través del editor integrado Javascript. Si bien este editor se ha desarrollado para permitir, entre otras funciones, incrustar imágenes (y objetos multimedia en general) de una forma intuitiva, es labor del sistema realizar un proceso de búsqueda de imágenes para asociarlas al texto. El proceso que se debe realizar, aunque a priori no sería necesario para componer la vista web, sí que lo es para generar la vista móvil del recorrido asociado. Adicionalmente, la aplicación para el responsable del museo muestra en el panel lateral, dentro de la pantalla de edición del contenido, la lista de objetos multimedia asociados actualmente al contenido, por lo que este proceso para asociar multimedia a contenidos se hace necesario de forma general.

El proceso de búsqueda de multimedia en un texto, o *string* en general, permite diversas y variadas soluciones. En nuestro caso hemos seguido una orientación basada en expresiones regulares, ya que con frecuencia son una técnica adecuada para el análisis sintáctico de textos. En Java el análisis mediante expresiones regulares se puede programar por medio de la combinación de las clases *Matcher* y *Pattern* dentro del paquete *java.util.regex*.

La clase *Pattern* es la representación compilada de una expresion regular (patrón), o lo que es lo mismo, representa a la expresion regular, que en el paquete java.util.regex necesita estar compilada. La clase *Matcher* es un tipo de objeto que se crea a partir de un patrón mediante la invocación del método *Pattern.matcher*. Este objeto es el que nos permite realizar operaciones sobre la secuencia de caracteres que queremos validar o en la secuencia de caracteres en la que queremos buscar. Por lo tanto, tenemos patrones que deben ser compilados, y a partir de éstos creamos objetos *Matcher* para poder realizar las operaciones sobre la cadena en cuestión:

```
public static List<String> buscarImgTags(String texto) throws Exception {
        List<String> imgTags = new ArrayList<String>();
        String patronImg = "<\\s*img[^>]*(/)?>";
        Matcher m = Pattern.compile(patronImg).matcher(texto);
        while(m.find()) {
                // procesar la imagen en busca de su atributo src
                [...]
        }
        return imgTags;
```

```
}
```

En el ejemplo anterior, además de la propia compilación del patrón y la iteración a lo largo de las coincidencias, hay que resaltar el formato usado para especificar la expresión regular (*patronImg*). En él, se ha indicado que se debe buscar el patrón compuesto por cero o más espacios (s*), seguido por ”img” (etiqueta HTML de imagen), y que se debe continuar haciendo concordar con cualquier secuencia hasta llegar al fin del nodo img, dado por ”>” o por ”/>”.

10.2.2. Marshalling XML: Castor

Para la generación de las respuestas en XML tanto para el servicio de feeds RSS como para la vista XML de un recurso siguiendo Dublin Core, se ha empleado Castor. El componente Castor es un framework de enlace de datos de código abierto para Java. Es, básicamente, el camino mas corto entre objetos Java, documentos XML, y tablas SQL. En el presente proyecto su uso ha sido específicamente para devolver un documento XML bien formado a partir de una clase Java del modelo.

Para conseguirlo, aunque Castor nos brinda facilidades se hace necesario escribir un archivo XML de correspondencia entre las propiedades del objeto y el documento XML que queremos obtener. Sin embargo, a partir de ello el código necesario para su generación se torna mucho más directo y simple de cara a su mantenimiento y reutilización. Podemos ver un ejemplo descriptivo de cómo se integraría Castor en Virtual Museum. Por una parte tenemos el propio archivo de correspondencias entre las propiedades y los atributos del XML a generar:

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping>
        <class name="com.museum4j.modelo.Obra">
                <map-to xml="dublinCore"/>

                <field name="title" type="java.util.Set" lazy="false" collection="set">
                        <bind-xml name="title" node="element" />
                </field>

                <field name="contributor" type="java.util.Set" lazy="false" collection="set">
                        <bind-xml name="contributor" node="element" />
                </field>

                <field name="creator" type="java.util.Set" lazy="false" collection="set">
                        <bind-xml name="creator" node="element" />
                </field>
                [...]
        </class>
</mapping>
```

Como se muestra, tenemos por ejemplo que el campo (*field*) *title* es un conjunto y que deberá generar una serie de elementos XML denominados ”title”. De forma equivalente, se va procediendo para el resto de atributos Dublin Core de la obra. Una vez que se han descrito las correspondencias, generar el documento XML conlleva muchas menos líneas que si hubiésemos diseñado una solución específica:

```
// Clase  com.museum4j.spring.vistas.VistaXML;
protected Node createDomNode(...) {
        String rutaMapping = "mappingDublinCore.xml";
        Mapping mapping = new Mapping();
        mapping.loadMapping(rutaMapping);
```

```
        // Obtener implementacion DOM
        DocumentBuilderFactory factory =  DocumentBuilderFactory.newInstance();
        // a partir del factory crear un document-loader
        DocumentBuilder loader = factory.newDocumentBuilder();
        // y crear el nuevo documento DOM
        Document doc = loader.newDocument();
        Marshaller marshaller = new Marshaller(doc);
        marshaller.setMapping(mapping);
        marshaller.marshal(obra);
        return doc;
}
```

Aunque el código anterior no es completo, el objetivo del mismo es hacer notar que la generación del XML a partir del objeto *obra* se hacer prácticamente en la penúltima línea.

10.2.3. Generación de la vista móvil: StringTemplate

Para la composición y generación de la aplicación *standalone* Java para móviles se ha optado por utilizar un enfoque similar al de plantillas Tiles o Velocity usadas en la vista web. El empleo de plantillas para generar la aplicación nos ha permitido enfocarnos a cada actividad concreta y no mezclar la labor de diseño de pantallas con la propia lógica de navegación y negocio presente en la aplicación.

Como ya se ha descrito en capítulos anteriores, la aplicación móvil para el visitante presenta un esquema común de pantallas compuesto por la presentación, pantalla introductoria del propio recorrido y una serie de pantallas por cada parada que constituyen el recorrido. Así, tomando un enfoque de plantillas, nuestra aplicación presenta tres plantillas bien diferenciadas, que incluyen algunas variables determinadas por el recorrido para el que se está generando la aplicación. De esta forma, una de las tareas a resolver en este apartado ha consistido en generar un archivo de código Java compilable a partir de una plantilla dada. De entre las diversas alternativas con que contábamos, hemos optado por el uso de StringTemplate.

StringTemplate es un motor de plantillas escrito en Java (aunque también se ha portado a C# y Python) para la generación de código fuente, páginas web, emails, o cualquier otra salida de texto formateado. Su característica más diferenciadora es que promueve de forma estricta la separación entre el modelo y la vista, enfoque que en Virtual Museum resulta idóneo.

Para la generación de código fuente Java a partir de plantillas StringTemplate basta con definir una plantilla:

```
public class Parada$parada.obra.idContenido$Form extends WDialog
{
        public void initDialog()
        {
                append("$texto.titulo$");
                append("$texto.descripcion$");
                $imagenes: { imagen |
                // Imagen $imagen.idMultimedia$
                Image imagen$imagen.idMultimedia$;
                try {
                        imagen$imagen.idMultimedia$ = Image.createImage("/$imagen.nombreFichero$");
                [...]
                }$
        [...]
```

Donde el texto encerrado entre $ denota variables y la construcción $imagenes: { imagen | ... }$ produce un bucle que itera por todos los objetos de la lista *imagenes*. A partir del diseño de esta plantilla, lo que deberíamos hacer en el código que genera el *jar* con la aplicación para el móvil, es indicar la plantilla a usar y pasarle como atributos las variables que va a usar:

```
// Clase com.museum4j.negocio.MovilManagerImp
StringTemplate codigoForm = grupo.getInstanceOf("paradaFormBase"); // nombre fichero plantilla .st
codigoForm.setAttribute("texto", texto);
codigoForm.setAttribute("parada", parada);
codigoForm.setAttribute("imagenes", texto.getImagenes());
return codigoForm.toString();
```

Justamente, esta orientación nos permite desacoplar la vista de la lógica necesaria para su composición, produciendo un código más limpio, fácil de seguir y por tanto, más mantenible y escalable.

10.2.4. Flickr

Flickr es un servicio de almacenamiento de fotografías que se ha hecho muy popular por la facilidad con la que se pueden organizar, subir, compartir las fotografías y también manejar vídeos. En Flickr se aúnan por primera vez en una aplicación varias innovaciones tecnológicas con nuevas tendencias sociales surgidas al calor de éstas, creando un así un espacio único en la red, en el que los usuarios pueden interactuar entre sí de diversas formas.

Flickr cuenta con una API disponible a los desarrolladores que la utilicen de forma ”no comercial” y en caso de que se desee realizar algo comercial es necesario que se realice un acuerdo previo para que sea posible. Según la descripción general oficial de la API para realizar una acción se debe seleccionar una llamada, enviar una solicitud, especificar un métodos y varios parámetros para recibir una respuesta con formato.

En nuestro caso, vamos a necesitar tres funcionalidades básicas:

- Enviar imagen
- Eliminar imagen
- Buscar una imagen dentro de Virtual Museum para determinar si ya está subida a Flickr.

Sería deseable contar con la opción de poder crear álbumes o colecciones pero esta función sólo está disponible para las cuentas ”pro”. Por eso, y para dar una solución aproximada cada imagen que se envie tendrá un título compuesto por el nombre del múseo más el propio nombre de la imagen.

Para las funciones de que disponemos se pueden usar tres formatos para el envío de solicitudes: REST, XML-RPC o SOAP. Dentro de estas opciones, vamos a optar por usar REST ya que a priori es el más simple y además contamos con un el *wrapper flickr4j* que nos ofrece las llamadas necesarias para nuestro fin.

De todas formas, este *wrapper* sólo lo vamos a usar para implementar los métodos de buscar y borrar, ya que para enviar una imagen Flickr no ofrece la vía normal porque según se comenta en su documentación requiere el envío de archivos binarios. Para subir una imagen también existen

varias alternativas y de entre ellas hemos elegido enviar un correo electrónico adjuntando la imagen e indicando en el título el nombre deseado que le queremos dar.

10.2.5. Picasa

Google dispone de una serie de APIs de datos (de forma abreviada ”GData”) que constituyen un sencillo protocolo estándar para leer e introducir datos en la Web. Estas interfaces utilizan dos formatos de sindicación estándar basados en XML: Atom o RSS. También cuentan con un sistema de publicación de feeds que incluye el protocolo de publicación Atom además de algunas extensiones para gestionar consultas (mediante el modelo de extensión estándar de Atom). Dentro de este conjunto de APIs que permiten manejar múltiples aplicaciones web Google (*Calendar, Documents, Blogger, etc*) también se encuentran los álbumes web de *Picasa*.

Con el API de Picasa, es posible acceder a los álbumes, fotos, comentarios y etiquetas que tengamos almacenados en nuestra cuenta. Para comunicarse con Picasa, nuestra aplicación ha desarrollado llamadas a los *feeds* correspondientes para:

- Seleccionar una dirección URL de un feed apropiado
- Hacer una solicitud a este feed.
- Crear un nuevo álbum
- Enviar una imagen nueva
- Buscar una imagen existente
- Borrar una imagen existente

10.3. Configuración del núcleo

10.3.1. Integración Struts/Spring

El patrón de diseño fundamental del sistema es MVC por lo que Struts representa uno de los pilares principales sobre el que que conectar el modelo de datos con la vista. Sin embargo la inyección de dependencia que nos da Spring resulta muy atractiva por lo que se ha optado por hacer que coexistan los dos. Aunque Spring también nos da un *framework* MVC, Struts sigue resultando mejor aproximación.

Las ventajas de integrar un sistema Struts en la plataforma Spring son diversas. La primera de todas es que Spring se diseñó de forma explícita para resolver algunos de los problemas de la ”vida real JEE”, como la complejidad, baja eficiencia, testabilidad, etc.

Para integrar Struts en Spring se pueden usar tres técnicas diferentes:

- Usar la clase Spring *ActionSupport*.

- Anular la clase Struts *RequestProcessor* con *DelegatingRequestProcessor* de Spring.
- Delegar la gestión de acciones Struts al entorno de Spring.

De las tres técnicas la primera puede resultar la más intuitiva ya que se basa en hacer que nuestras acciones deriven directamente de la clase *org.springframework.web.struts.ActionSupport* de Spring en vez de las propias clases de acción de Struts. Sin embargo, produce un fuerte acoplamiento de la acción Struts a la plataforma Spring; si se decide reemplazar Spring, habría que reescribir el código. Además, como la acción Struts no está bajo el control de Spring, no puede aprovecharse de sus beneficios.

La segunda opción desacopla las acciones Struts de Spring ya que nuestras acciones siguen derivando de clases de acción Struts pero se registrarían como *beans* en el archivo de configuración de Spring, así como en el archivo de configuración de Struts. Al definir también las acciones en Spring, éstas se pueden aprovechar de la inversión de control. Respecto a la primera técnica, ésta supone una mejora ya que nuestras acciones serían inconscientes de la existencia de Spring, por lo que se podría intercambiar Spring por otro contenedor de inversión de control sin tener que replantear nuestro código Struts. Sin embargo, esta aproximación también presentaría algunos problemas. Si se usase un procesador diferente de peticiones, habría que integrar la clase Spring *DelegatingRequestProcessor* manualmente. El código así aumentado supondría un factor de complejidad añadido y podría reducir la flexibilidad de la aplicación de cara al futuro.

Una mejor aproximación a la integración de Struts con Spring es delegar la gestión de acciones Struts al *framework* Spring. Para conseguirlo, se puede registrar un *proxy* en el archivo de configuración de acciones de Struts. El *proxy* será responsable de buscar la acción Struts en el contexto Spring. Como la acción está bajo el control de Spring, puede poblar la propiedades *JavaBean* de las acciones, y dejar la puerta abierta por si se quisiesen aplicar otras características de Spring como la orientación a aspectos.

Sin embargo para la aplicación de usuario de Virtual Museum esta solución no resultaba totalmente apropiada por lo que en este caso, pasamos a usar otra combinación de componentes. En esta aplicación nos desligamos de Struts ya que desde el punto de vista de acciones, en este caso vamos a contar con un par de acciones. Sin embargo, sí que seguimos interesados en acceder a los *beans* de la lógica de negocio y que éstos accedan los objetos del modelo de la misma forma en que lo hacían las otras dos aplicaciones. Por eso, aunque esta aplicación no comparte Struts como despachador de peticiones, y se deja únicamente en manos de Spring, sí que se siguen compartiendo los componentes de las capas inferiores.

10.4. Velocity

Las plantillas usan la extensión *.vm*, como las acciones disponibles por el servlet Struts. Así se consigue de una forma transparente que éstas no sean accesibles directamente por el usuario. Es por eso que si un usuario sube un archivo de macros Velocity con esta extensión al gestor multimedia y la enlaza en algún contenido, al dirigirnos al enlace obtendremos un aviso de que la acción no se encuentra, con la correspondiente traza en los logs del sistema.

10.5. Construcción y Despliegue

10.5.1. Apache y Tomcat

Aunque Tomcat puede ejercer sin problemas de servidor web no está tan optimizado para ello como Apache. Por ello, nuestro objetivo es que las peticiones a Apache se redirijan a Tomcat, para lo que hemos utilizado el conector *mod_jk*, desarrollado por Jakarta.

10.5.2. Documentación del código

La documentación del código es un aspecto fundamental como información general del sistema, y es un factor clave a la hora de mantener el código. Durante la implementación de los distintos paquetes del proyecto, además de los propios comentarios del lenguaje Java, se ha empleado la guía propuesta por la herramienta *Javadoc*. El uso de estas convenciones ha aportado un valor añadido al proyecto, ya que de esta forma, se ha podido generar una documentación HTML, de forma automática, del código del sistema en general, y además, permite que los IDEs de programación (como *Eclipse* o *NetBeans*) autcompleten según va escribiendo el programador.

10.5.3. Sistema de Trazas

Un sistema de trazas es una herramienta importante en una aplicación, ya que durante las iteraciones de construcción permite al programador disponer de una vía de depuración del código, mientras que cuando el sistema está en producción, añade información que no se presenta al usuario de lo que va ocurriendo en el sistema, lo cual resulta de particular interés, incluso como medio de auditoría. En el proyecto, la herramienta que hemos usado como sistema de trazas es Log4j.

Log4j es una API, que forma parte del proyecto Jakarta de Apache, para manejar el registro (*log*) de operaciones en nuestros programas. Esta API proporciona mecanismos para configurar dónde y cómo mostrar los mensajes, el tipo y mensaje a mostrar, y la fuente de datos, por lo que la convierte en una opción estupenda para Virtual Museum.

10.5.4. Compilación

Para la construcción si bien tenemos un amplio conjunto de ficheros de código fuente fácilmente reconocible por cualquier IDE para que se pueda compilar, hemos optado por usar la herramienta **ant** en la que a partir de un archivo de configuración podemos construir, limpiar o bien generar la documentación del código de una forma sencilla y sin dependencias de terceros programas. Para esta construcción también podíamos haber usado maven pero aunque esta herramienta es más potente y cada vez se usa más en los proyectos, ant nos daba todas las funcionalidades que necesitaba nuestro sistema.

A continuación se muestra un fragmento del fichero *build.xml* que permite la construcción del proyecto. En él, cabe hacerse notar el uso de XML para la descripción de tareas, la dependencia de

tareas, y el uso de variables para poder parametrizar el proceso:

```
<project name="museum4j" default="ayuda" basedir="./">

  <!-- propiedades globales del proyecto -->
  <property name="fuente" value="src" />
  <property name="destino" value="classes" />
  <property name="jars" value="jars" />
  <property name="lib" value="lib" />
  <property name="doc" value="doc" />
  <property name="dist" value="dist" />

<!--  ********************        TAREAS         ***********************  -->
  <target name="todo" depends="compilar, documentar, aplicacion">
  </target>

<!--  *********************   TAREA LIMPIAR   **************************  -->
  <target name="limpiar">
    <!--
       Eliminamos los directorios de la vez anterior.
       Eliminamos los dos juntos para no favorecer que
       existan versiones de documentacion que no se
       corresponden con las clases compiladas y viceversa
    -->
    <delete dir="${destino}" />
    <delete dir="${doc}" />
  </target>


<!--  *********************   TAREA COMPILAR  **************************** -->
  <target name="compilar" depends="limpiar">
   <!--
         Creamos el directorio si no existe. Y no existira porque
         lo hemos eliminado con el objetivo limpiar, del cual
         depende compilar
    -->
    <mkdir dir="${destino}" />
    <!-- Copiar los Mapeos de Hibernate *.hbm.xml -->
    <copy todir="${destino}/com/museum4j/hibernate" verbose="false">
          <fileset dir="${fuente}/hibernateMappings" includes="*.xml" />
    </copy>
    <!-- Copiar los archivos de propiedades:
                    - ApplicationResources: Textos para soporte i18n
                    - commons-logging: Configuracion para Jakarta-Commons-Logging
                    - log4j: Configuracion para Jakarta-Log4j
    -->
    <copy todir="${destino}" verbose="false">
          <fileset dir="${fuente}" includes="*.properties" />
    </copy>
    <!-- Servicios Web XFire -->
    <mkdir dir="${destino}/META-INF" />
    <mkdir dir="${destino}/META-INF/xfire" />
    <copy todir="${destino}/META-INF/xfire" verbose="false">
          <fileset dir="${fuente}" includes="services.xml" />
    </copy>
    <!-- Tipos Mime de Activation -->
    <copy todir="${destino}/META-INF" verbose="false">
          <fileset dir="${fuente}" includes="mime.types" />
    </copy>
    <!-- Compilacion  -->
    <javac srcdir="${fuente}" destdir="${destino}" deprecation="off" nowarn="off"/>
  </target>
```

10.5.5. Métricas

Como se ha comentado previamente, se han obtenido diversas métricas a partir de las herramientas PMD, CPD y *Findbugs*, lo que nos ha llevado a correcciones y mejoras en iteraciones posteriores. La herramienta *Findbugs* dispone de una GUI directamente así que su uso era inmediato, mientras que por parte de PMD y CPD su empleo ha sido a través de tareas ant que producen informes html o texto plano directamente.

10.6. Pruebas

Durante la fase de pruebas se verifica el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema a ser entregadas al cliente. Hay que tener en cuenta que confirmar que una aplicación está probada completamente es una tarea que normalmente no se puede asegurar de forma general, así que el objetivo principal de esta etapa será plantear y llevar a cabo una serie de casos de pruebas con el fin de minimizar la aparición de errores.

La estrategia seguida ha sido partir de los casos de uso, y para cada caso de uso, realizar pruebas en cada escenario distinto. Además de ello, también se le añadido al rol del equipo de trabajo Tester la tarea de, sin haber usado antes el sistema, intentar llevar a cabo actividades propias del responsable del museo para evaluar y supervisar los posibles problemas de uso que se le presentaban. En ambas líneas de pruebas los resultados han sido satisfactorios, a pesar de que en algunas iteraciones las pruebas daban con la aparición de errores, se ha ido corrigiendo en las posteriores.

Capítulo 11

CONCLUSIONES

Un buen libro es aquel que se abre con expectación y se cierra con provecho.

Louise Alcott

Índice del Capítulo

11.1. Introducción

Como parte final de la presente memoria, se presenta una evaluación para ciertas funciones del sistema desarrollado. En general, se puede decir que se han cumplido los objetivos que se proponían al comenzar el proyecto, produciendo una aplicación completa que puede responder a una demanda real en el mercado. Las diversas ideas planteadas a continuación se han tratado de organizar en función del aspecto de la aplicación a que hacen referencia.

11.2. Diseño

El sistema resultante del desarrollo tiene funcionalidades enmarcardas habitualmente dentro de los sistemas de gestión de contenidos (CMS), y dentro de este apartado se ha producido una aplicación bastante intuitiva y versátil, que puede dar lugar a multitud de sitios web de diversa índole.

Desde el punto de vista de diseño, la personalización de la apariencia a través de de plantillas es muy potente, (máximo cuando el diseñador emplea estilos CSS), lo cual da lugar a museos virtuales con diseños muy diversos.

11.3. Servicios

Además de las funcionalidades propias de un gestor de contenidos, el proyecto aporta varias funcionalidades que le dan un valor añadido respecto a otros, como son la interconexión con otros sistemas (Flickr, Picasa o Google Maps), y el acceso a la información no sólo presentando una vista web, sino habilitando un acceso a otras aplicaciones mediante servicios web, XML y RSS, e incluso ofreciendo ciertos contenidos vía móvil. Con todo, el sistema construído presenta amplias funcionalidades, sobre todo en el aspecto Web 2.0, al añadir mecanismos de comunicación entre los distintos usuarios, dando un enfoque más colaborativo al resultado.

Sin embargo, en el apartado de interconexión con otros sistemas, se puede indicar que el uso de las APIs, servicios web, o medios en general que ofrecen son particulares para cada aplicación. Particularmente hablando, la comunicación con Google Maps no ha resultado tan flexible como cabía esperar al principio del proyecto. Primeramente, hay que advertir que este sistema proporciona una vía simple para integrar mapas en nuestro sitio web, pero se nos presenta costoso el publicarlos en Google Maps, ya que sigue un esquema de indexado similar al del búscador ”clásico” Google de palabras. Si bien, el posicionamiento de Virtual Museum en Google puede ser relativamente fácil, no ocurre así dentro de Google Maps.

11.4. Dublin Core

Cabe destacarse por encima de los servicios de valor añadido que presenta el proyecto, el empleo del conjunto de elementos Dublin Core. En este punto, lo primero que se debe indicar es que la propia inclusión de un estándar de metadatos aproxima el proyecto hacia la Web Semántica, y en particular, Dublin Core compone un estándar simple pero flexible, que puede extenderse fácilmente lo que ha propiciado que su adopción no obligue a que los museos virtuales generados estén estrictamente ligados al arte.

11.5. Arquitectura

Desde el punto de vista de la arquitectura elaborada para la implementación, hay que hacer énfasis en la estructuración por capas, produciendo un sistema escalable que facilita el intercambio de los componentes usados en cada capa.

En concreto, la combinación de tecnologías Struts, Spring e Hibernate, particularmente, presentan una seria alternativa en desarrollos Java para este tipo de aplicaciones. En esta línea, primeramente hay que agredecérselo a a la existencia de soluciones libres en entornos Java, que no son tan habitua-

les, ni ofrecen tal abánico de alternativas, en otros entornos. Y no sólo por la simple existencia sino por la potencia que ofrecen y que en la práctica son empleadas para construir sistemas empresariales.

Por otra parte, el uso de frameworks como Struts y Spring es particularmente interesante por el enfoque que hacen en cuanto a la vista guiándose por acciones. Esta aproximación ofrece una clara separación entre el controlador y la vista. Así, esta última puede ser cualquier tipo de componente capaz de *renderizar* una respuesta para el modelo dado. Como se ha visto el proyecto hace uso de diversas tecnologías como JSP, Velocity, Tiles, XML, etc. Si bien este enfoque por acciones para la lógica de presentación en una aplicación web ha sido apropiado por la estructuración y robustez que provee, en algunas ocasiones su mantenimiento (el archivo de configuración *struts-config.xml*), puede resultar tedioso por los problemas que origina mantener esta lógica de forma declarativa en un archivo XML de relativa magnitud.

11.6. Vista móvil

Dentro de la funcionalidad propia de la vista móvil del sitio web generado para el museo virtual, merece la pena comentar el enfoque adoptado. Durante el desarrollo del proyecto, hemos asistido a la aparición de nuevas tecnologías para la construcción de aplicaciones en dispostivios móviles, ahora que tienen mayores capacidades a todos los niveles (físicas, hardware, conectividad, etc). Es por ello, que en algunos momentos del desarrollo se planteó la posibilidad de rehacer la aplicación generada para móviles, dejando apartado el desarrollo basado en la configuración CLDC y perfil MIDP.

Las aplicaciones construidas bajo el perfil MIDP están orientadas a dispositivos con capacidades reducidas (32Kb de memoria en ejecución para la pila Java, entrada de datos alfanumérica reducida, pantallas de al menos 96x54 píxeles, baja potencia computacional, etc), fácilmente alcanzables hoy en día. Por una parte, este tipo de aplicaciones J2ME presenta la virtud de no presentar grandes requerimientos por lo que esta limitación puede volverse una ventaja ya que permitiría la ejecución de la aplicación en un conjunto potencialmente mayor de dipositivos.

Capítulo 12

LÍNEAS FUTURAS DE TRABAJO

En lo pasado está la historia del futuro.

Juan Donoso Cortés

Índice del Capítulo

12.1. Introducción

Después de comentar las conclusiones del trabajo realizado, llega la hora de proponer nuevos requisitos o líneas de desarrollo que en cierto modo pueden resultar interesantes, al menos como reflexión del futuro que podría esperarse de este proyecto.

12.2. Bibliotecas digitales y Dublin Core

Las bibliotecas digitales y las publicaciones electrónicas ya están presentes en el día a día. No son un concepto académico para debatir o un sueño utópico. Existen muchas interpretaciones y opiniones de la situación actual, pero una de ellas podría ser que las bibliotecas digitales están al final de una fase inicial y empezando una nueva. Esta primera fase puede verse como un movimiento de las publicaciones tradicionales y las colecciones de las bibliotecas a las redes digitales. Fundamentalmente, se trata de un mejora de los medios de información establecidos mediante el uso de estas nuevas tecnologías. Podría pensarse que, en esta línea, la siguiente fase verá nuevos tipos de colecciones y servicios que no tienen su análogo en los medios tradicionales.

Desde el punto de vista particular de Dublin Core, Virtual Museum ha explorado ciertas capacidades en las que se pone de manifiesto su tremendo potencial. Sin embargo, en el futuro y según avance la Web, podría pensarse en explotar más la funcionalidad que ofrece la aplicación en cuanto a su soporte de Dublin Core, de cara a ofrecer otras sintaxis como microformatos por ejemplo, o de disponer de los cualificadores para cada uno de los términos básicos. Además, merecería la pena plantearse la incorporación de otros estándares de metadatos, y en cierta medida, la inclusión de cruces (*crosswalks*) automáticos de Dublin Core al resto de estándares.

12.3. Virtual Museum

Ligado a la evolución de las museos virtuales, resultaría interesante que este proyecto la acompañará con ampliaciones y nuevos enfoques. Cabe destacar que las ideas plasmadas en este apartado referente a la aplicación y su diseño son producto principalmente de considerar que el sistema necesita ampliarse por un uso mayor y más amplio. Por eso, si bien en las decisiones de diseño tomadas para el presente proyecto se tenía muy en cuenta ofrecer una aplicación intuitiva y de fácil uso, las futuras ampliaciones podrían deberse a la incorporación de funciones más avanzadas.

Lo primero que puede echarse en falta en el presente proyecto es un gestión más sofisticada de usuarios. Suele ser relativamente normal que en los sistemas de publicación de contenidos exista cierta jerarquía de roles a la hora de establecer permisos sobre las acciones de enviar contenidos, editar, publicar, eliminar, etc.

De igual forma que para los contenidos se podría contemplar una organización de roles, también podría plantearse una estructura jerárquica de contenidos, permitiendo en primer lugar que el conjunto de categorías no estuviesen predefinidos, y a la vez que se pudiese plantear una estructura flexible de varios niveles de profundidad. Dicho de otro modo, esta organización flexible podría verse como la posibilidad de organizar los contenidos en carpetas y subcarpetas.

De la misma forma, que se puede pensar en relajar la organización de contenidos, éstos por sí mismos también podrían volverse más elaborados, manteniendo un atributo de estado que podría indicar si el contenido está propuesto, publicado, borrado (lógico), pendiente de revisión, etc. Esto, aunque haría más pronunciada la curva de aprendizaje, podría enriquecer el sistema a la hora de mantener contenidos cuando el museo virtual contase con una cantidad considerable.

Como consecuencia de los puntos anteriores, si el sistema cuenta con más información las estadísticas deberían evolucionar para aprovechar estos avances. Incluso sin tener estas ampliaciones, podría ser atractivo mantener el sistema de estadísticas como un módulo más independiente del resto de funcionalidades, y basándose en soluciones genéricas de análisis web como *AWStats* o *Webalizer*.

Al hilo de la separación del sistema de estadísticas del núcleo de la aplicación, la orientación a módulos o *plugins* es un aspecto cada vez más presente en sistemas similares en cierto sentido a Virtual Museum. El desarrollo de módulos permitiría abrir el sistema a una comunidad no sólo de usuarios sino de desarrolladores que podrían ampliar la funcionalidad básica del núcleo, de una forma parecida a lo que ya ofrece al motor de plantillas.

Desde el punto de vista de la aplicación para dispositivos móviles, como se comentaba en el apartado anterior, se ha enfocado en generar una aplicación que sea capaz de ejecutarse en terminales

modestos en cuanto a prestaciones. Sin embargo, las capacidades crecientes de los dispositivos actualmente supera ampliamente el perfil MIDP por lo que podría plantearse que el sistema fuera capaz de generar una aplicación más rica en cuanto a diseño y experiencia. En este sentido, últimamente están apareciendo (o cobrando más valor) nuevas tecnologías que auguran dicho riqueza visual y funcional, como *Android, Betavine, Gears* etc., o sistemas operativos propios que ya existían en el mercado, como *Palm OS, Windows Mobile, Apple OS X for portables, Symbian o RIM*. A pesar de ello, el presente proyecto no optó por introducir estas tecnologías ya que esta proliferación de entornos y sistemas operativos para móviles hace que la compatibilidad sea un problema cada vez mayor, y sería deseable cierta uniformidad de cara a la comunidad de desarrolladores.

Y para acabar este apartado, pueden tenerse en cuenta también los criterios relativos a seguridad. Aunque el presente proyecto ha tenido en cuenta diversas consideraciones de seguridad en varios niveles de la aplicación, la puesta en funcionamiento de un sistema de estas características en un entorno de explotación real necesitaría un análisis mayor y pormenorizado de los posibles puntos débiles del sistema que podrían hacer que un atacante ”echara abajo” el sistema.

Parte III

Apéndices

Apéndice A

MANUALES DE USUARIO

Índice del Capítulo

A.1. Manual del administrador

Para poder entrar a la aplicación se debe proporcionar un usuario y clave de administrador (figura A.1).

A.1.1. Introducción

La aplicación del administrador se plantea en un menú divido en los diversos apartados que conforman las entidades base del sistema.

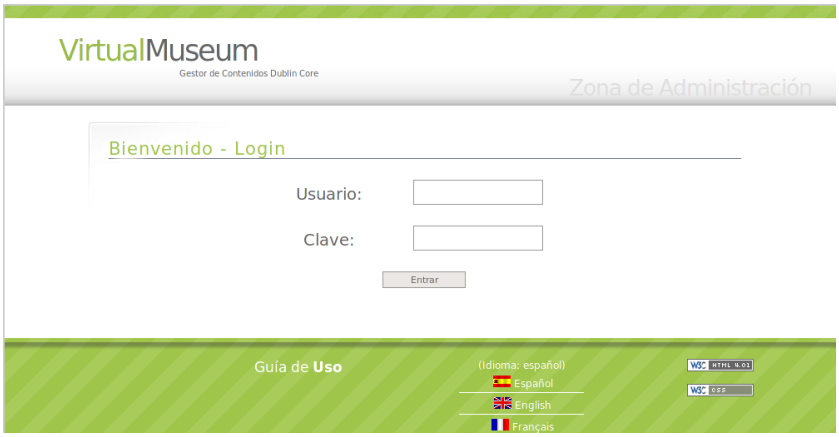
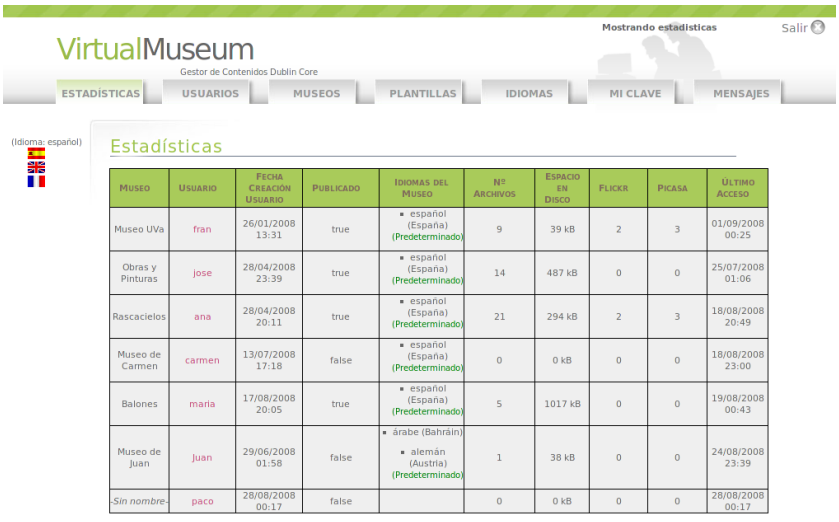


Figura A.1: Administrador - Login

A.1.2. Estadísticas

La pantalla de estadísticas resume distinta información para cada museo virtual que puede serle útil al administrador. Relacionadas con el usuario se tienen las fechas de creación del usuario y de la última vez que ha entrado, mientras que desde el punto de vista del museo propiamente dicho se muestra información acerca de los idiomas para los que está configurado, la ocupación en disco por los ficheros subidos, y de ellos cuántas imágenes están publicadas en Flickr y en Picasa.



Museo	Usuario	Fecha Creación Usuario	Publicado	Idiomas del Museo	Nº Archivos	Espacio en Disco	Flickr	Picasa	Último Acceso
Museo UVa	fran	26/01/2008 13:31	true	▪ español (España) (Predeterminado)	9	39 kB	2	3	01/09/2008 00:25
Obras y Pinturas	jose	28/04/2008 23:39	true	▪ español (España) (Predeterminado)	14	487 kB	0	0	25/07/2008 01:06
Rascacielos	ana	28/04/2008 20:11	true	▪ español (España) (Predeterminado)	21	294 kB	2	3	18/08/2008 20:49
Museo de Carmen	carmen	13/07/2008 17:18	false	▪ español (España) (Predeterminado)	0	0 kB	0	0	18/08/2008 23:00
Balones	maria	17/08/2008 20:05	true	▪ español (España) (Predeterminado)	5	1017 kB	0	0	19/08/2008 00:43
Museo de Juan	juan	29/06/2008 01:58	false	▪ árabe (Bahráin) ▪ alemán (Austria) (Predeterminado)	1	38 kB	0	0	24/08/2008 23:39
-Sin nombre-	paco	28/08/2008 00:17	false		0	0 kB	0	0	28/08/2008 00:17

Figura A.2: Administrador - Estadísticas

A.1.3. Usuarios

La sección de usuarios presenta inicialmente el listado general de usuarios responsables del museo. Cada usuario tiene al menos login y clave, siendo ambos campos obligatorios en la creación de un nuevo usuario. Adicionalmente, la información del usuario se completa con una serie de datos personales, de contacto y relacionados al museo como son: nombre y apellidos, correo electrónico, teléfono, dirección y ocupación. Estos campos adicionales son de carácter optativo y en cualquier caso, sólo serán visibles por el administrador.

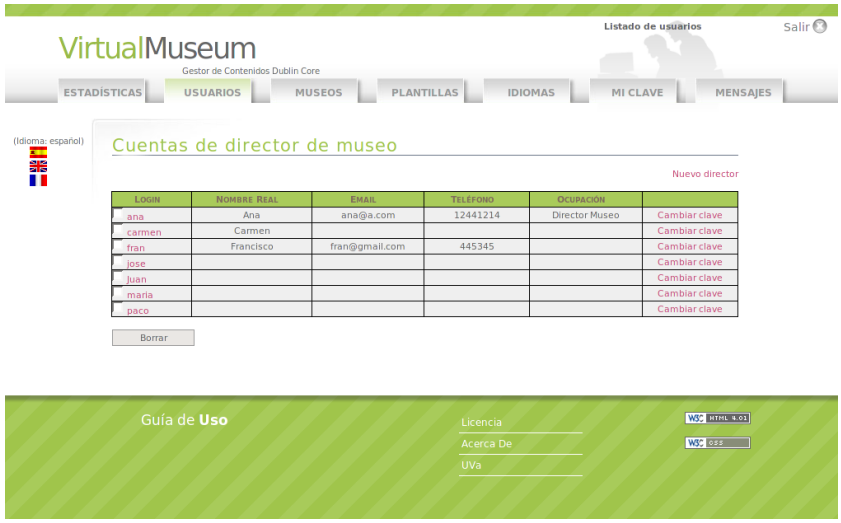


Figura A.3: Administrador - Usuarios (rol responsable del museo)

Durante la creación de un nuevo usuario, se deberá especificar la clave dos veces para evitar descuidos y asegurar que la clave es la deseada. Por comodidad, el sistema brinda la posibilidad de escoger una de las claves que propone por medio del enlace de generación de claves de forma aleatoria.

Una vez creado el nuevo usuario, éste aparecerá en el listado general. Para modificar los datos básicos del mismo, se puede pulsar directamente sobre el enlace del usuario, donde se mostrarán sus detalles, permitiendo la actualización de cualquiera de ellos, a excepción del login que es usado como identificador único de la cuenta.

El cambio de clave que usa un usuario para entrar a la aplicación del responsable del museo se realiza de forma similar, y en la que, como en creación, ha de escribirse dos veces la nueva clave que se desea. De la misma forma que en la creación, la aplicación ofrece la posibilidad de elegir una clave de las que propone.


Tenga en cuenta que un cambio de clave para un usuario dado producirá que no vuelva a poder entrar a su aplicación una vez que éste desconecte de su sesión actual, siendo necesaria la comunicación del cambio de clave al propio usuario.

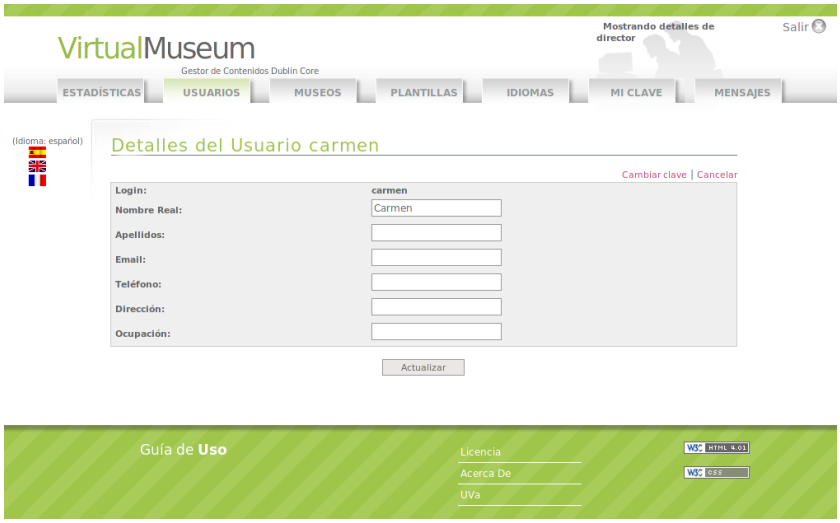


Figura A.4: Administrador - Detalle usuario

Figura A.5: Administrador - Cambiar clave de usuario

Además de la modificación de la información de un usuario existente, el sistema también permite su eliminación definitiva. Esta eliminación borrará el usuario y acarreará que se elimine también su museo virtual generado (si es que ya tenía uno asociado). Para eliminar un usuario (o más de uno) basta con seleccionarlo y pulsar sobre el botón ”Borrar”; el sistema pedirá confirmación de la operación que se está a punto de realizar, y en caso afirmativo se procederá con la eliminación definitiva del usuario.

A.1.4. Museos

La sección de museos presenta inicialmente el listado general de museos virtuales generados. Cada museo tiene al menos un nombre, plantilla y usuario asociado, siendo campos obligatorios en la creación de un nuevo museo. Adicionalmente, la información del museo se completa con una serie de datos a saber: dirección, localidad, teléfono, fax y correo electrónico. Estos campos adicionales

son de carácter optativo y en cualquier caso, sólo serán visibles por el administrador y por el propio usuario.

Figura A.6: Administrador - Museos Virtuales

Una vez creado el nuevo museo, éste aparecerá en el listado general. El administrador puede modificar los atributos del propio museo e incluso administrarlo. Como desde la aplicación del responsable del museo se pueden modificar los atributos básicos que se han especificado durante su creación, para simplificar la interfaz, el enlace del nombre del museo llevará a la gestión del museo virtual como si fuera el propio usuario responsable del museo.

Nótese que cualquier modificación efectuada dentro del museo virtual es inmediata y que la verá el usuario responsable del museo generado.

Además de la modificación de la información de un museo existente, el sistema también permite su eliminación definitiva. Para eliminar un museo (o más de uno) basta con seleccionarlo y pulsar sobre el botón ”Borrar”; el sistema pedirá confirmación de la operación que se está a punto de realizar, y en caso afirmativo se procederá con la eliminación definitiva del museo virtual.

A.1.5. Plantillas

Las plantillas son usadas para cambiar la apariencia general de los museos virtuales generados. Desde este apartado se pueden gestionar las distintas plantillas que estarán disponibles a los usuarios responsables del museo para que puedan personalizar su sitio. De todas formas, hay que hacer notar que aunque el usuario elija una plantilla, se puede personalizar aún más el sitio por medio de hojas de estilo CSS propias o bien personalizando la apariencia de los contenidos gracias al editor integrado.

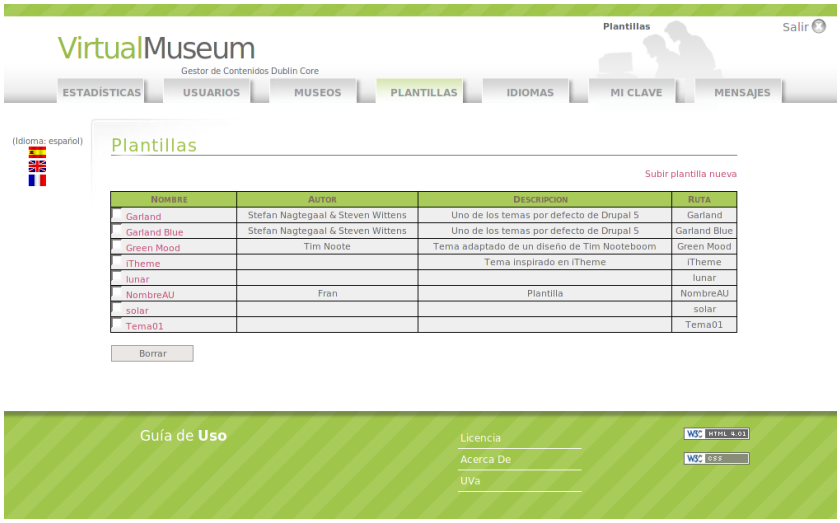


Figura A.7: Administrador - Plantillas

Importación de plantillas

El usuario administrador puede importar nuevas plantillas al sistema subiendo un archivo empaquetado **zip**, para evitar que se tenga que subir cada uno de los archivos individuales que componen la plantilla.

Una vez que se haya especificado el fichero zip que contiene la plantilla a importar, el sistema intentará descomprimir y validar que el fichero importado es realmente una plantilla válida en Virtual Museum. Los requerimientos que debe satisfacer un archivo empaquetado de plantilla se exponen a continuación.

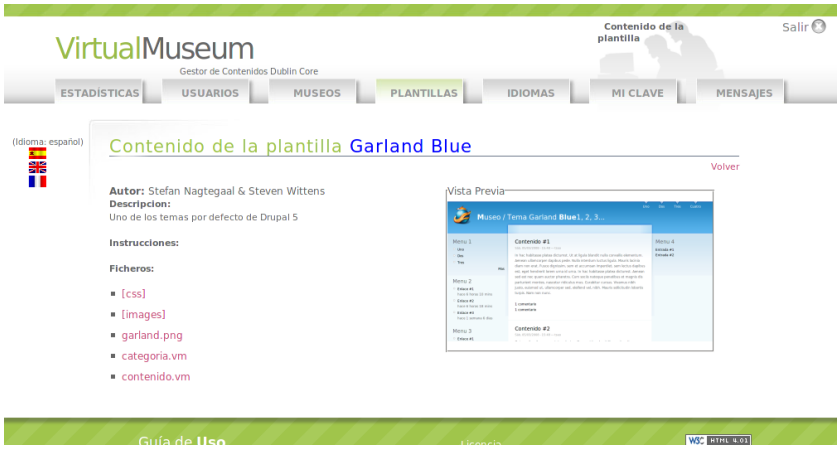


Figura A.8: Administrador - Ficheros de plantilla

Elementos de una plantilla

Cada plantilla está compuesta por un conjunto de archivos de los cuales algunos son opcionales. Este conjunto de archivos fijará la composición, apariencia y comportamiento del museo virtual generado. Primeramente, para que el sistema empiece a validar un archivo empaquetado recién subido, es obligatorio que éste contenga el descriptor de plantilla *museum4j.xml*. Este descriptor es un archivo XML que presenta la siguiente estructura:

```
<museum4j>
        <nombre>Mi primera plantilla</nombre>
        <autor>Francisco</autor>
        <vistaPrevia>miniatura.jpg</vistaPrevia>
        <descripcion>
                Esta plantilla es un ejemplo
        </descripcion>
        <instrucciones>
                Instrucciones de estilo adicionales para el correcto uso de esta plantilla
        </instrucciones>
</museum4j>
```

Los atributos del descriptor de plantilla tienen el siguiente significado:

- **nombre (Obligatorio):** El nombre que dará título a la plantilla.
- **autor:** El nombre del autor que la ha diseñado.
- **vistaPrevia:** Ruta relativa dentro de la plantilla referente al archivo de imagen que presenta una vista previa de la apariencia general. Si se especifica un fichero, éste deberá existir dentro de la plantilla.
- **descripcion:** Campo de texto libre con comentarios y descripciones varias que añaden alguna observación.
- **instrucciones:** Campo de texto libre con instrucciones adicionales que pueden ayudar al responsable del museo en el correcto uso de la plantilla dentro de su museo. Estas instrucciones con frecuencia se referirán a los nombres de menú que se han empleado y a los estilos CSS empleados.

Además de este fichero que describe los atributos propios de la plantilla, es necesario incluir dos ficheros que contienen la estructura y diseño del museo virtual generado con esta plantilla:

- **contenido.vm**: Fichero de plantilla HTML usado para *renderizar* contenidos.
- **categoria.vm**: Fichero de plantilla HTML usado para *renderizar* categorías completas.

Opcionalmente se pueden incluir ficheros de plantilla HTML para cada tipo de contenido y para la pantalla de envío de comentarios. En cuanto a los ficheros de plantilla por tipo de contenido los ficheros a incluir deberían ser nombrados según el nombre de la categoría, y seguidos por la extensión *.vm*:

- autor

- evento
- generico
- infoLocalizacion
- infoMuseo
- novedad
- obra
- recorrido

Para los archivos de plantilla de envío de comentarios, se han de incluir dos archivos denominados *nuevoComentarioForm.vm* y *comentarioOkForm.vm*. El primero de ellos se usará para mostrar un cuadro de texto donde escribir el comentario, mientras que el segundo informará al visitante de que el comentario ha sido enviado.

Dependiendo de cómo se muestre la categoría habrá ocasiones en que ésta se muestra presentando una introducción al texto completo dejando el contenido completo en otra como enlace a otra página. Esta página puede llevar un diseño definido por la propia plantilla, aunque si no se especifica Virtual Museum usará uno por defecto.

Una vez vistos los ficheros necesarios de una plantilla usable en Virtual Museum, quedarían por incluir aquellos ficheros multimedia (CSS, imágenes, javascript, flash, etc) propios de los ficheros de plantilla que se han usado en el propio diseño y generación de la plantilla.

Motor de plantillas

El motor de plantillas que usa Virtual Museum intenta combinar la facilidad de uso junto con la flexibilidad en el diseño. De esta forma, para la composición de los ficheros de plantilla se puede hacer uso de un conjunto de etiquetas que generarán html automáticamente y que servirán de apoyo en el diseño de las plantillas. La referencia completa de las etiquetas que hay disponibles se puede consultar en el apéndice C.

A.1.6. Idiomas

Esta pantalla lista todos los idiomas disponibles que maneja el sistema. Estos idiomas serán utilizados por los usuarios responsables del museopara poder configurar un museo virtual en distintos idiomas y así completar los textos para los contenidos en los distintos idiomas que elijan.

A.1.7. Mi Clave

Desde esta pantalla se puede cambiar la clave propia que ha utilizado el usuario administrador para entrar a la aplicación. Para asegurar que el cambio de clave se realiza correctamente, la aplica-

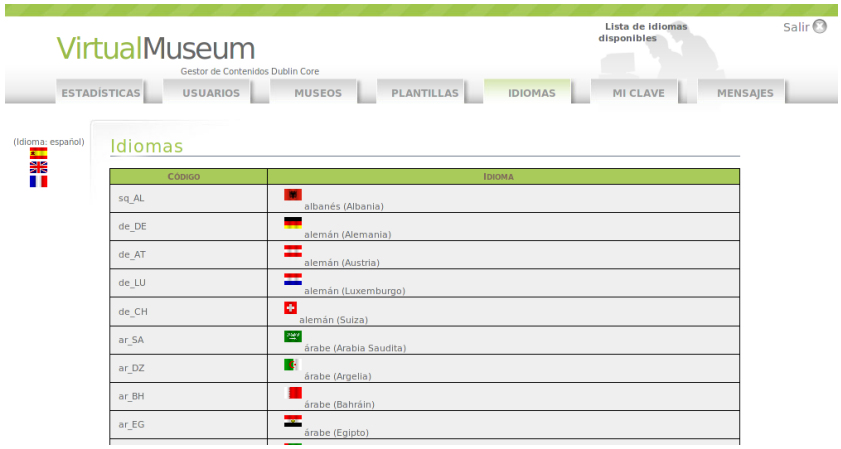


Figura A.9: Administrador - Idiomas

ción pedirá que se indique la clave dos veces y así tratar de evitar desaciertos o descuidos. Ambos campos son obligatorios para que el cambio de clave se realice efectivamente.

Figura A.10: Administrador - Cambio de clave para el administrador

El sistema recién instalado establece un usuario administrador y clave por defecto que inicialmente son:

- ***Usuario:*** *admin*
- ***Clave:*** *12345*

Para ofrecer una vía alternativa a la elección de clave, el sistema proporciona un servicio de generación de claves de forma aleatoria, mediante el que la aplicación irá generando claves compuestas

por números y letras de longitud ocho, con el objetivo de aumentar la seguridad del sistema y disminuir el riesgo potencial por claves de relativa baja complejidad.

A.1.8. Mensajes

Los usuarios de cada museo pueden enviar mensajes al administrador con peticiones de nuevos servicios con los que les gustaría contar o bien hacer observaciones constructivas de la aplicación del responsable del museo. Desde esta pantalla el administrador verá listados todos los mensajes que se han ido mandando a lo largo del tiempo, junto con el usuario y fecha en que se envió.

A.1.9. Cambio de Idioma

La aplicación puede cambiar el idioma en que se visualiza por medio del selector de idioma que aparece en el panel izquierdo de la pantalla.

Esta selección de idioma es distinta que la del sitio web generado para cada museo virtual ya que aquí los textos para cada idioma han sido configurados previamente, mientras que para el museo virtual generado es el usuario responsable del museo el que edita el texto para cada idioma de los que ha elegido tener disponible su sitio web.

Figura A.11: Administrador - Idiomas

A.2. Manual del responsable del museo

A.2.1. Introducción

Virtual Museum es un gestor de contenidos (CMS) orientado a la elaboración de museos virtuales. Ayuda a crear y mantener de una forma sencilla y directa las colecciones de obras y ponerlas a disposición del público en general. Además, promueve la interoperabilidad con otros sistemas ofreciendo en primer lugar soporte Dublin Core y adicionalmente RSS, Google Maps, Flickr, Picasa, recorridos para moviles, etc.

Figura A.12: Responsable del museo - Login

Como gestor de contenidos consiste en una interfaz que controla la base de datos donde se alojan los contenidos del sitio. El sistema permite manejar de manera independiente el contenido y el diseño. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio sin tener que darle formato al contenido de nuevo, además de permitir la fácil y controlada publicación del sitio.

A.2.2. Panorámica de la aplicación

La aplicación del usuario responsable del museo plantea una estructura basada en menús desplegables que agrupan los grandes bloques funcionales de la aplicación.

A grandes rasgos se puede decir que Virtual Museum se divide en los siguientes grandes bloques:

Figura A.13: Responsable del museo - Menú principal

- Idiomas en los que se va a estar disponible el museo virtual generado.
- Contenidos editados por el usuario.
- Colección de obras y sus autores y recorridos asociados.
- Eventos y novedades informativas de la actualidad del museo.
- Gestor multimedia que permite la incorporación de objetos multimedia en general (imágenes, estilos CSS, javascript, flash, vídeos, etc) para la descarga o uso dentro del diseño del museo.
- Publicación del museo virtual para que quede disponible al público general bien vía web a través de un navegador o bien vía móvil en el que se podrá acceder a los recorridos escogidos a través de un dispositivo móvil.
- Servicios Web 2.0 que permiten la interoperabilidad y colaboración con otros sistemas y aplicaciones. Entre los servicios ofrecidos se pueden encontrar Flickr, Picasa, Google Maps, acceso a los contenidos por medio de RSS o servicio web, mensajes del visitante al responsable del museo, inclusión de publicidad, etc.
- Dublin Core como estándar de metadatos asociado a los contenidos en general y particularmente enfocado a la colección de obras.
- Diseño y composición que posibilita la personalización de la apariencia del museo virtual generado gracias al armado de menús, elección de plantilla e inclusión de estilos y objetos multimedia adicionales propios del sitio.
- Estadísticas para ofrecer información al usuario responsable del museo información acerca de sus propios contenidos y los accesos que ha realizado el visitante en general.

A.2.3. Museo Virtual

Museo

Permite actualizar los datos del museo que representan informaciones generales del museo ”físico”, tales como: Nombre del museo, dirección, localidad, teléfono, fax y correo electrónico. De los campos anteriores, se recomienda no dejar vacío el nombre del museo al menos, ya se utilizará en el sitio web del museo virtual generado.

Figura A.14: Responsable del museo - Menú Museo Virtual

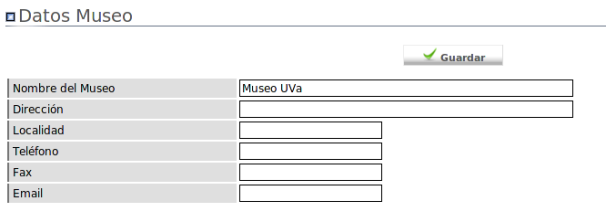


Figura A.15: Responsable del museo - Museo

Idiomas

Cada museo virtual puede ser configurado de forma que el visitante pueda elegir el idioma, dentro de los definidos, en el que desea visualizar los contenidos del sitio. Para que esto sea posible, primero se deben escoger los idiomas que se van a desarrollar en contenidos. Un museo recién creado no tendrá ningún idioma por defecto por lo que será una de las primeras tareas a completar. Un museo virtual debe tener al menos un idioma predeterminado ya que cada contenido presentará textos en un idioma, y el visitante por defecto entrará al sitio web en el idioma seleccionado por defecto. El idioma predeterminado se resalta en amarillo y puede ser modificado a posteriori.

Opcionalmente, se pueden añadir otros idiomas de forma que por cada idioma incorporado la aplicación habilita nuevos campos dentro de cada contenido para editar el texto asociado al idioma recién añadido. Debe tenerse en cuenta que por cada idioma añadido es labor del usuario responsable del museo editar y mantener los textos por cada contenido.

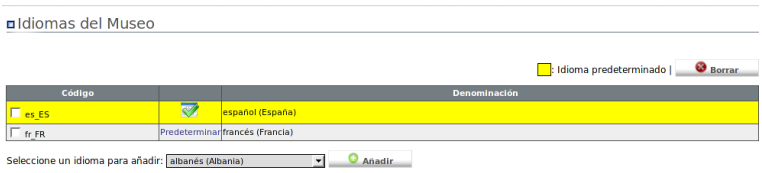


Figura A.16: Responsable del museo - Configuración de idiomas

Además de poder añadir otros idiomas, puede ocurrir que ocasionalmente no se necesite un idio-

ma de los que se habían configurado. En este caso, se puede borrar seleccionándolo (uno o varios) y pulsando el botón ”Borrar”. Debe hacerse notar con especial énfasis que la eliminación de un idioma provocará que se eliminen los textos asociados a ese idioma para todos los contenidos del museo, por lo que se aconseja asegurarse de que el idioma elegido para borrar ya no va a usarse en el futuro.

Gestor Multimedia

Dentro de la sección de Gestor Multimedia el usuario responsable del museo puede administrar los archivos multimedia subidos a Virtual Museum y que por regla general le servirán para adjuntarlos a contenidos o bien utilizarlos dentro del diseño global del sitio web. Será habitual por tanto, subir archivos como imágenes y vídeos para incorporarlos dentro de contenidos, ofrecerlos por descarga al visitante, o bien añadir hojas de estilo CSS para refinar el diseño del sitio web a partir de la plantilla elegida.

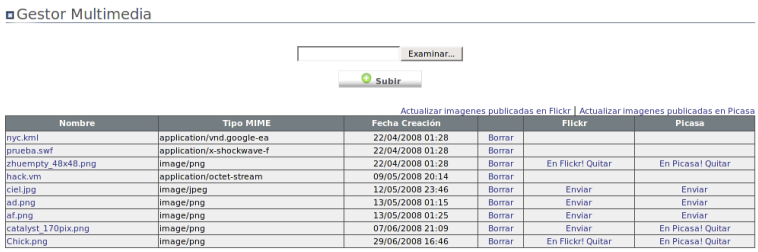


Figura A.17: Responsable del museo - Gestor Multimedia

Para subir un nuevo archivo a Virtual Museum basta con pulsar el botón ”Examinar”, navegar por la estructura de carpetas local para localizar el archivo que se desea subir, y a continuación pulsar el botón ”Subir”. El archivo subido (después del tiempo que tarde en transferirse al servidor) aparecerá en el listado general de archivos. Cada archivo subido muestra además de su nombre, el tipo MIME que describe su formato y la fecha en que se ha subido a Virtual Museum. Asimismo, se muestra información relacionada con Flickr y Picasa pero ambas columnas serán tratadas más adelante.

Cada archivo que se ha subido puede verse con tal de pulsar en él, ya que el título del mismo conduce a su contenido. Cabe destacarse en este punto, que los archivos de hojas de estilo CSS son un tipo especial de archivo ya que refinan el diseño del sitio web y agregan ciertas modificaciones al diseño establecido por la plantilla elegida. Es por ello que para estos archivos se habilita justo a continuación del título un enlace que permite editar su contenido directamente *online* y no tener que editarlo en local y volverlo a subir.

Cuando un archivo ya no se use para ningún fin dentro del museo, pueden eliminarse definitivamente pulsando el enlace ”Borrar”. Nótese que si el archivo estaba siendo usado en algún contenido, éste desaparecerá del mismo.

Comentarios

Dentro de los servicios brindados al público general, el museo virtual también ofrece al visitante la posibilidad de dejar comentarios generales con opiniones, sugerencias o críticas de la visita realizada al museo virtual. Cada comentario vertido se podrá consultar desde este apartado de comentarios, en el que además del propio comentario se muestra la fecha y dirección IP del mismo.

Publicar

Un museo virtual no estará disponible vía web por el visitante hasta que sea publicado. Una vez que se haya completado contenidos suficientes para el usuario, se puede acceder a esta zona para publicar el museo y hacerlo accesible al visitante por medio de un navegador web.

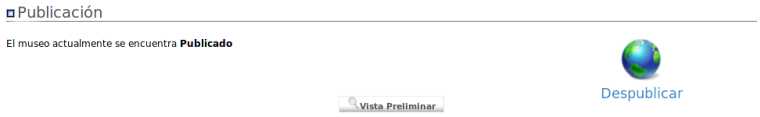


Figura A.18: Responsable del museo - Publicación vista web

Por defecto, un museo recién creado no está publicado ya que inicialmente está vacío y no daría información al visitante. La publicación es un proceso rápido que afecta a todos los contenidos del museo, ya que éstos no disponen de un atributo de estado o similar para publicar o no individualmente cada contenido. Cabe mencionar en este punto que no es necesario despublicar y publicar de nuevo el museo cada vez que se modifique un contenido, ya que los cambios aplicados son aplicados inmediatamente al sitio web del museo virtual generado.

Para visualizar el sitio web del museo virtual generado también se puede pulsar en el icono de lupa situado a la derecha del nombre del museo.

Vista Móvil

De la misma forma que un museo puede publicarse para hacerlo accesible vía web, también se pueden generar vistas para móviles relativamente menores con alguna parte del museo completo. Una vista para móviles del museo virtual se basa por una parte en un recorrido ya creado y marcado para móviles, y por otra parte en un idioma de los configurados para el museo. La vista para móviles genera una aplicación Java (*jar*) que puede ser ejecutada en dispositivos móviles con soporte Java. Debido a las limitaciones de los dispositivos móviles la aplicación generada, la vista móvil contendrá la información asociada a un recorrido y un idioma. De esta forma, se minimiza el tamaño total del archivo generado y además, se pueden generar tantas vistas móviles como recorridos e idiomas se tengan, y componer así una serie de guías específicas fácilmente descargables individualmente a modo de folleto particular del museo virtual.

La vista móvil que genera Virtual Museum presentará inicialmente una pantalla de bienvenida, seguida por el texto del propio recorrido, y tantas pantallas como paradas tenga el recorrido. Los

textos tanto del recorrido como de cada una de las pantallas pueden incluir código HTML por la inclusión de estilos y objetos multimedias, por lo que previa a la generación de las pantallas se lleva a cabo un proceso de ”limpieza” del texto para poder componer así un texto plano que omite el código HTML y añade al final del texto las distintas imágenes asociadas al contenido.

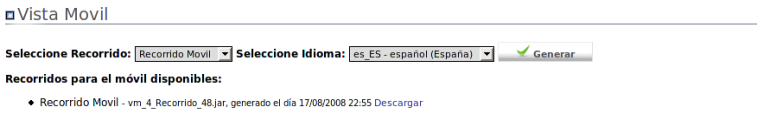


Figura A.19: Responsable del museo - Publicación vista para dispositivos móviles

Una vez que se ha generado una vista móvil para un recorrido dado, éste aparecerá en la lista de ”Recorridos para el móvil disponibles”, donde se podrá descargar para su distribución o incluso importarlo como un archivo dentro del Gestor Multimedia y enlazarlo dentro de un contenido.

Si se elige volver a generar una vista móvil para un recorrido e idioma que ha sido generado previamente, la aplicación sobreescribirá automáticamente la vista antigua.

A.2.4. Contenidos

En Virtual Museum existe una jerarquía de organización del contenido basada en dos niveles:

- Categorías: contenedores por tipo de contenido.
- Contenidos: textos y multimedias asociados editados por el usuario responsable del museo.

Todo contenido pertenece a una categoría, o dicho de otro modo, tiene un tipo de contenido. El conjunto de categorías viene predefinido por el sistema y no se podrá variar. Tanto la sección (en el menú) de colección, eventos y ésta misma se refieren a los distintos tipos de contenidos que ofrece Virtual Museum, a saber:

- Información del Museo
- Información de ubicación o localización
- Contenido general
- Obra
- Obra Maestra
- Autor
- Recorrido
- Evento

- Novedad

Para ver los contenidos de un tipo dado se puede utilizar el enlace correspondiente del menú; para ver todos los contenidos del museo independientemente de su categoría se puede usar el enlace ”Ver todo” dentro del menú ”Contenidos”. En cualquier caso, se obtiene un listado de los contenidos pertinentes en forma de tabla, en la que además del título se muestra su fecha de creación, última modificación, categoría, estilo CSS usado y número de visitas o *hits* que han realizado los visitantes a dicho contenido.

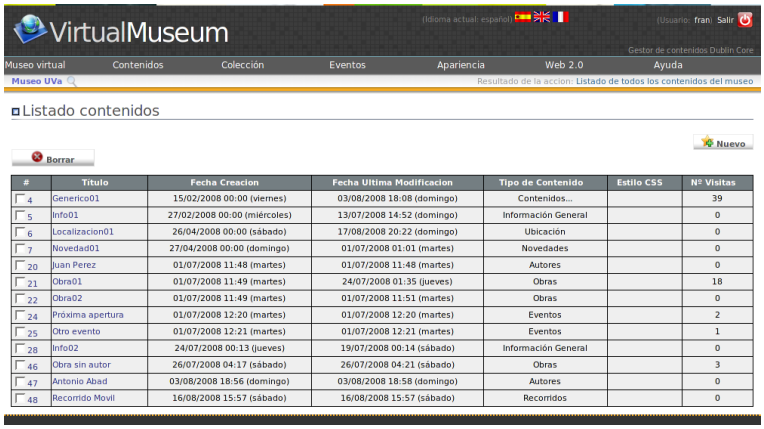


Figura A.20: Responsable del museo - Contenidos

Cada contenido puede ser editado pulsando sobre el enlace de su título, y además puede ser borrado con tal de seleccionarlo y pulsar el botón ”Borrar”. Un contenido borrado no puede ser recuperado mediante la aplicación, por lo que se invita al usuario responsable del museo a asegurarse de aquellos contenidos que desea eliminar definitivamente.

Cuando se pulsa sobre el enlace de un contenido, se muestran los detalles del mismo permitiendo su edición. Además de los propios textos del contenido, la aplicación da información en el panel lateral acerca de las fechas en que se ha creado, modificado, la última vez que ha sido visitado y el número de veces. En este mismo panel lateral, se puede indicar un clase CSS para personalizar el estilo general del contenido.

Todos los contenidos presentan una serie de atributos modificables comunes como son el estilo CSS, y por cada idioma, su título y texto. Los contenidos de las categorías Información del Museo, Ubicación, General y Novedad contemplan estos campos y no añaden ninguno más. Sin embargo, el resto de categorías agregan algunos atributos propios. A continuación, se enumeran los atributos que añaden las categorías de Autor y Evento:

- **Autores**: Nombre y apellidos, fecha de nacimiento, fallecimiento, corriente artística y país.
- **Eventos de calendario**: fecha de inicio, fin, activo y contenido al que referencia.

Los atributos de las obras, obras maestras y recorridos se ven a continuación.

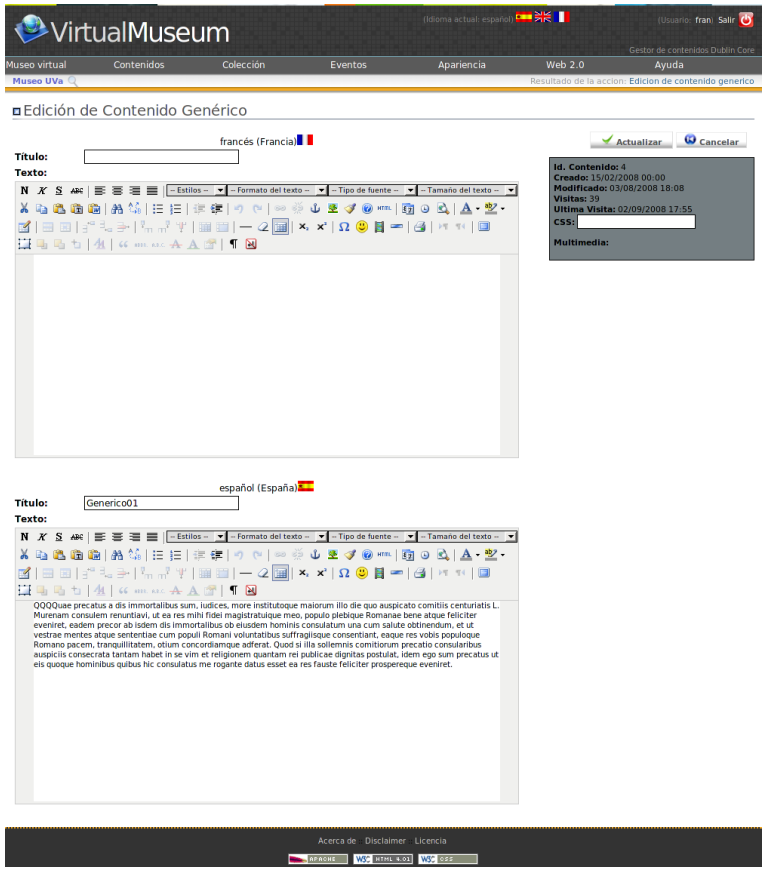


Figura A.21: Responsable del museo - Detalle contenido

Obras y Dublin Core

Los contenidos de la categoría Obras componen el eje central de contenidos dentro de Virtual Museum. En primer lugar esta categoría añade el autor de la misma que puede ser uno de los incluidos de la categoría Autor, aunque si la obra es anónima o directamente no es aplicable el campo de autor, se puede especificar ”ninguno”.

Además del campo de autor, una obra es el único contenido susceptible de seguir completamente el conjunto de elementos de Dublin Core. De esta manera, la edición de una obra propone los quince elementos propios del conjunto básico de Dublin Core. En general, los elementos de Dublin Core son repetibles por lo que Virtual Museum permite indicar más de un valor por cada atributo para la mayoría de ellos, excluyendo a los atributos *type, format y date*. Si bien la aplicación muestra en forma de solapas cada uno de los atributos Dublin Core, el usuario podrá percatarse que no se han especificado los atributos *relation* y *description*; esto es debido a que el primero se rellena automáticamente por medio de los objetos multimedia incluidos dentro del contenido, y que el atributo *description* por su parte, se rellena, de forma análoga, gracias a los textos por cada idioma.

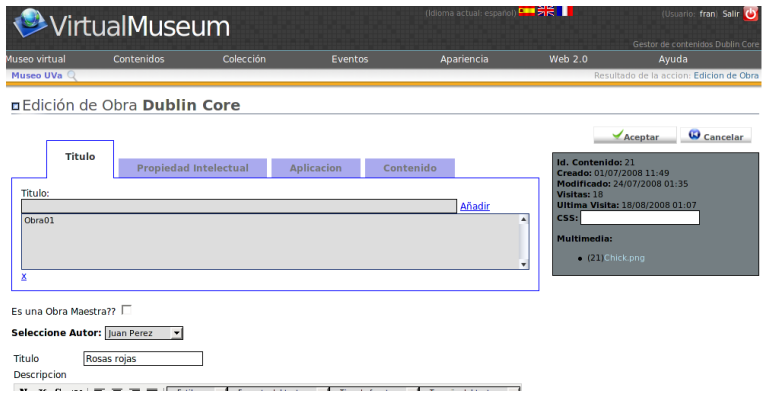


Figura A.22: Responsable del museo - Detalle obra

Recorridos

A partir de las obras existentes, el usuario responsable del museo puede crear recorridos a lo largo de las distintas obras que por su naturaleza, relación, etc son adecuadas para que el público las visite de forma ordenada. Es por ello que un recorrido se basa en un contenido genérico (al incluir título y textos) al que añade un conjunto de paradas.

Cada parada se refiere a una de las obras existentes, de forma que el recorrido puede componer una secuencia ordenada de obras.

Textos

Cada contenido está compuesto por uno o más textos y por otros datos característicos de su categoría o generales. Por cada idioma escogido desde el apartado A.2.3 de configuración de idiomas aparecerá un editor que permitirá escribir el texto asociado a ese idioma. Dicho editor permite además de escribir el texto para cada idioma, incluir otros elementos hipertextuales como objetos multimedia, enlaces, etc, y todo ello con un estilo directo e intuitivo.

Los textos por idioma asociados a un contenido no están relacionados entre sí de modo en que el usuario puede escribir libremente el texto deseado.

Editor de Textos

El editor de textos de Virtual Museum entra en la categoría de editores WYSIWYG (*what you see is what you get* - "lo que ves es lo que obtienes"), que permiten escribir un documento viendo directamente el resultado final. El editor brindado por Virtual Museum ofrece un amplio abánico de posibilidades que permiten ajustar el diseño del contenido e incrustar objetos hipertextuales. Dentro de esta última funcionalidad se pueden encontrar:

- Insertar enlaces a contenidos y objetos multimedia o bien direcciones externas al museo virtual.

- Incrustar imágenes  y objetos multimedia  provenientes del Gestor Multimedia.

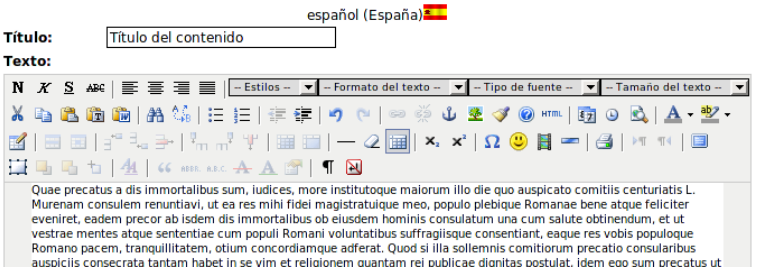


Figura A.23: Responsable del museo - Editor de textos

Los enlaces a elementos dentro del sitio del museo virtual pueden ser otros contenidos o bien objetos multimedia con tipo MIME *application*. Por otra parte, las imágenes y objetos multimedia que se pueden incrustar son los archivos de tipo MIME *image* y *application*, respectivamente.

Al incluir objetos multimedia y guardar el contenido, el sistema analizará cada uno de los textos del contenido en busca de potenciales objetos propios el Gestor Multimedia, para guardar la asociación y así, en la siguiente ocasión que se edite el contenido aparecerán listados en el menú lateral.

Respecto a las funciones que comprende el editor relativas a diseño, hay que tener en cuenta que permiten un nivel de personalización al detalle y que implican un refinamiento sobre el diseño especificado en primer lugar por la plantilla, y en segundo lugar por el estilo CSS dado para el contenido general.

A.2.5. Apariencia

Si bien las secciones anteriores daban acceso a la configuración de los fondos del museo, la sección Apariencia del menú engloba distintos aspectos relacionados con la forma, y están destinados exclusivamente a la visualización web del museo virtual.

Página inicial

Dentro de todos los contenidos que el usuario responsable del museo ha creado ha de elegirse uno de ellos para que sea la página inicial del sitio web, para que dé la entrada al museo virtual.

Menús

Los menús en Virtual Museum están formados por items referentes a contenidos o categorías. Si el item del menú se refiere a un contenido, en el sitio web se mostrará dicho contenido directamente, mientras que si es una categoría el sitio mostrará, por regla general, el conjunto de los contenidos para el tipo dado dentro del museo.

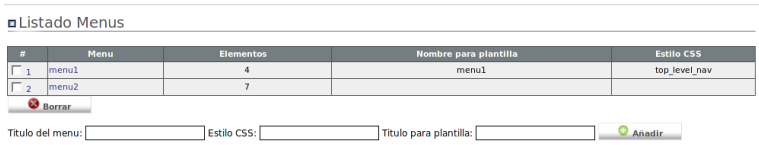


Figura A.24: Responsable del museo - Configuración de menús

Los menús componen un punto de conexión entre el propio museo virtual y la plantilla escogida. Un menú, además de su conjunto de items, tiene un estilo CSS, título propio y título para plantilla. El estilo CSS puede venir especificado por la plantilla o bien aparecer definido en algún archivo de hojas de estilo CSS añadido desde el Gestor Multimedia. Además, tiene dos títulos: uno usado por el usuario responsable del museo para referirse a él dentro de su aplicación, y otro usado para hacerse corresponder con el incluido en los archivos de la plantilla.

Figura A.25: Responsable del museo - Configuración de los elementos del menú

El conjunto de items en Virtual Museum no presenta profundidad por motivos de sencillez, de forma que para un item del menú no puede especificarse un subitem. Sin embargo, se podrían añadir tantos menús como los previstos por la plantilla.

Plantillas

El motor de plantillas de Virtual Museum constituye un aspecto fundamental en la personalización del sitio web para el museo virtual generado, y es que gracias a las plantillas la apariencia del sitio puede variar profundamente.

Lo primero que hay que hacer notar es que una plantilla no sólo afecta a la forma y colocación

de los distintos elementos dentro de la página sino que puede presentar características de comportamiento propias.

La elección de plantilla por parte del usuario responsable del museo es rápida e intuitiva: basta con elegir una de la lista de vistas previas

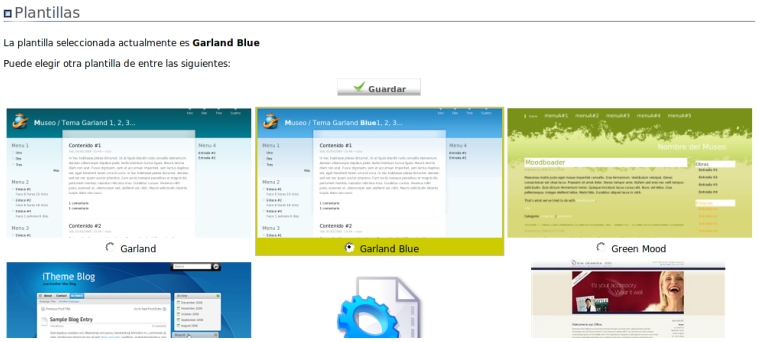


Figura A.26: Responsable del museo - Plantillas

A.2.6. Web 2.0

Los servicios de interoperabilidad que aporta Virtual Museum pueden englobarse en parte bajo los principios de la llamada Web 2.0 por su orientación a la colaboración e intercambio ágil de información entre usuarios y/o otros sistemas.

Google Maps

La funcionalidad que recoge Virtual Museum sobre Google Maps se basa en proporcionar inicialmente la localización física del museo. Normalmente, se indicarán los campos de longitud y latitud, aunque eventualmente puede darse el caso de necesitar completar la localización con el campo de altura.

Figura A.27: Responsable del museo - Localizacion física para Google Maps

Una vez que se ha rellenado la información de localización física, se pueden visualizar los mapas asociados a dichas coordenadas. Virtual Museum provee dos medios de representación: el primero (figura A.28) se basa en la API de Google Maps para incrustar un mapa interactivo dentro de nuestro sitio web, de forma que su código fuente podría ser usado para incorporarlo dentro de un contenido propio.

La segunda vía (figura A.29) consiste en obtener el mismo mapa que en el caso anterior pero en vez de tener un mapa (*javascript*) interativo, se muestra una imagen (normalmente en formato *gif*),

Figura A.28: Responsable del museo - Mapa interactivo

de forma que en algunos casos puede ser más conveniente su uso.

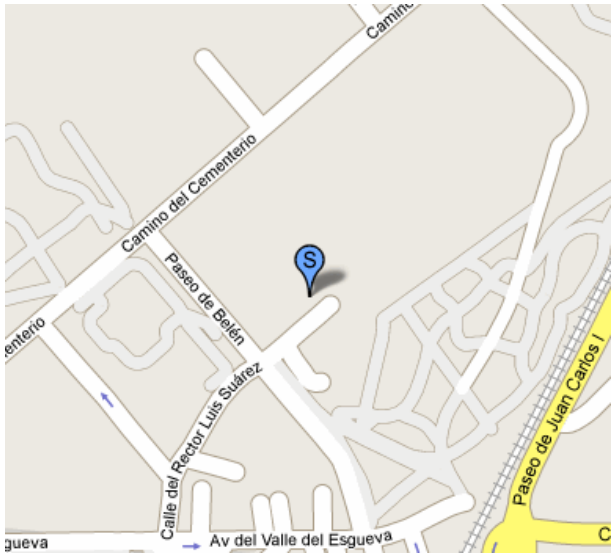


Figura A.29: Responsable del museo - Mapa como imagen

Publicidad

Dentro de esta sección, se puede escribir una porción de código HTML para que se incluya en el sitio web generado (figura A.30). Dicho código HTML puede venir dado por sistemas ajenos de publicidad y anuncios tales como Google AdSense.

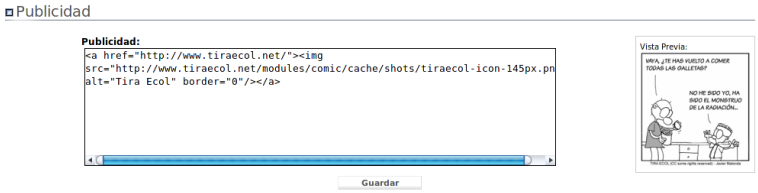


Figura A.30: Responsable del museo - Publicidad

Dublin Core

La sección de Dublin Core recoge dos servicios que permiten exponer, dentro del sitio web generado para el museo, cierta información Dublin Core. El primer servicio habilita la posibilidad de que el museo virtual permita acceder a los atributos de una obra en formato XML o bien mediante etiquetas *meta* incrustadas dentro del propio HTML. Y por otra parte, el segundo servicio plantea la opción de activar un servicio web a través del cual otras aplicaciones independientes a Virtual Museum puedan acceder a información Dublin Core acerca de los contenidos mantenidos por el sistema.

RSS

El servicio de feeds RSS (figura A.31) ofrecido por Virtual Museum permite la sindicación de los contenidos que se han modificado durante el último mes, en la línea de los habituales portales de noticias y blogs.

Figura A.31: Responsable del museo - Configuración RSS

Los feeds RSS generados presentan un título y descripción, que vienen dados por el propio contenido. Hay que hacer notar que la descripción, al proceder del texto, puede incluir código HTML por la inclusión de estilos y objetos multimedias, por lo que previa a la generación de feeds se lleva a cabo un proceso de ”limpieza” del texto para poder componer así un texto plano.

Otro servicio?

Si dentro de los servicios ofrecidos por Virtual Museum el usuario no encuentra alguno que le resultaría útil, o tiene algún otro tipo de sugerencia sobre la experiencia de uso del sistema, la aplicación le abre un vía sencilla de comunicación (figura A.32) con el administración para hacerle llegar las distintas ideas que tenga. Si bien Virtual Museum ofrece diversos servicios de comunicación

con los visitantes y otros sistemas, no está de más que también provea de un mecanismo para la comunicación usuario responsable del museo - administrador.

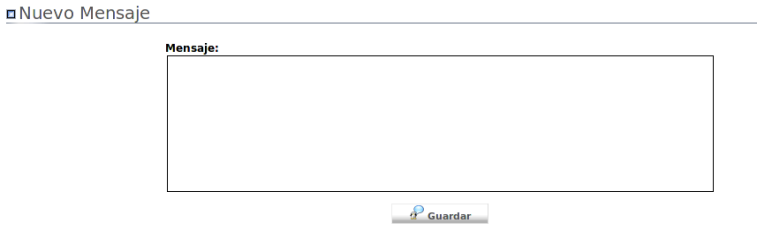


Figura A.32: Responsable del museo - Enviar mensaje al administrador

Apéndice B

MANUAL DE INSTALACIÓN

B.1. Instalación de Virtual Museum

B.1.1. Requisitos Mínimos Software

Para instalar Virtual Museum el proceso de instalación parte de unos requisitos previos o un software mínimo que necesita estar instalado:

- Java 2 Standard Developer Kit (JDK 1.6). *http://java.sun.com*
- Contenedor JSP: Tomcat de la rama 6. *http://tomcat.apache.org*
- MIDP 2.0. *http://java.sun.com*
- MySQL (se recomienda usar al menos la versión 4). *http://www.mysql.com*

El presente manual no cubre la instalación de los componentes anteriores, ya que el usuario normalmente encontrará información abundante en el sitio web propio del proveedor de cada tecnología. Sin embargo, puede ser de utilidad hacer notar que para la correcta ejecución de la aplicación deben estar configuradas ciertas variables del sistema:

- JAVA_HOME (JDK)
- CATALINA_HOME (Tomcat)
- MIDP_HOME (MIDP)

De ellas, habitualmente una instalación típica les dará valor coherentemente. Aún así, merece la pena hacer énfasis en que la variable PATH debería contener el directorio bin de la JDK y el directorio bin de MIDP, ya que la aplicación necesitará de ambas rutas para poder generar la aplicación para dispositivos móviles, al compilar y desplegar ”al vuelo” el código para cada recorrido.

B.1.2. Proceso de Instalación

Para el proceso de instalación en máquinas Linux se propone un *shell script* con una interfaz de usuario gráfica, consiguiendo una experiencia más amena.

Los pasos de la instalación están estructurados en forma de asistente, como las habituales instalaciones de otros sistemas, a saber:

1. Bienvenida a la instalación
2. Se busca una máquina virtual Java (sobre la que se ejecutará el contenedor Tomcat)
3. Se busca una instalación de Tomcat, para desplegar la aplicación
4. Creación y poblado inicial de las tablas MySQL para la puesta en marcha del sistema. El usuario usado no tendrá porqué ser el mismo (por cuestiones de seguridad entre otras) que el usado para la ejecución de la aplicación.
5. Confirmación de la inicialización de datos en MySQL
6. Configuración del usuario MySQL usado por la propia aplicación: se abrirá el editor predeterminado del usuario para abrir el archivo de configuración
7. Se le pide al usuario que reinicie la aplicación dentro de Tomcat para que la nueva configuración surta efecto
8. Agradecimientos por usar Virtual Museum

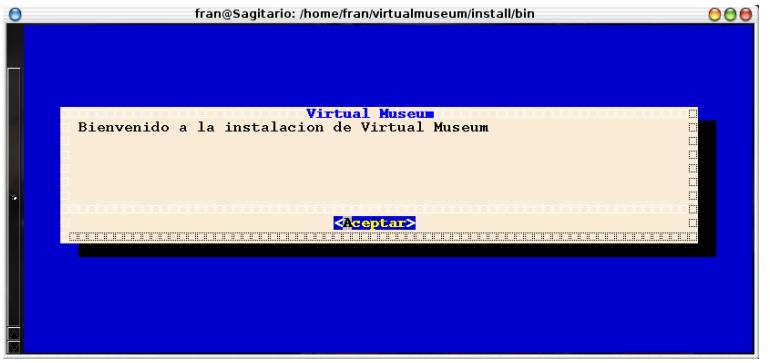


Figura B.1: Paso 1: Bienvenida a la instalación

Figura B.2: Paso 2: Búsqueda de la máquina virtual Java

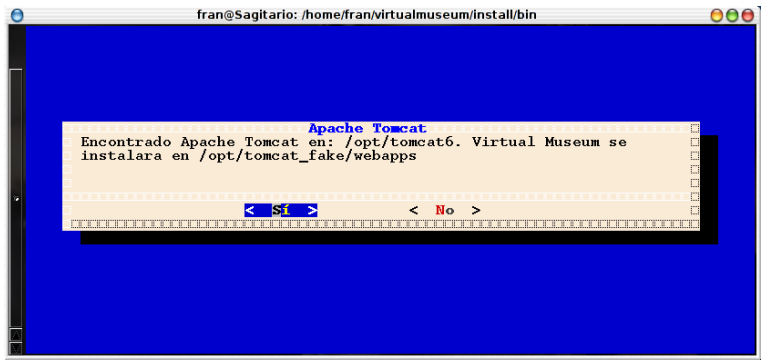


Figura B.3: Paso 3: Búsqueda del contenedor web Tomcat

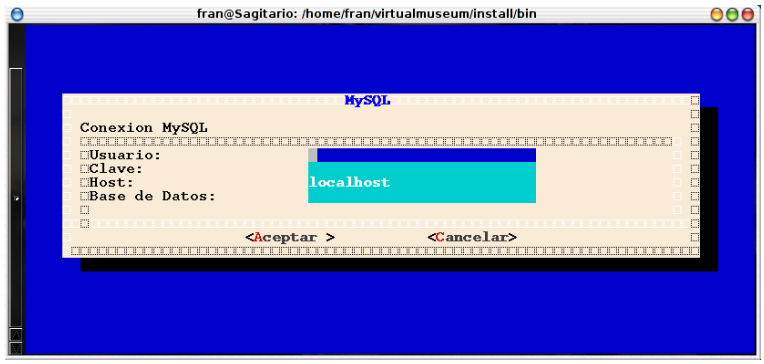


Figura B.4: Paso 4: Configuración de la conexión MySQL para la creación e inicialización de tablas

Figura B.5: Paso 5: Confirmación de la ejecución correcta de los scripts SQL

Figura B.6: Paso 6: Configuración de la conexión MySQL para la aplicación Virtual Museum

Figura B.7: Paso 7: Reinicio de la aplicación Virtual Museum, para cargar la configuración MySQL

Figura B.8: Paso 8: Finalización de la instalación

Apéndice C

REFERENCIA DE ETIQUETAS EN PLANTILLAS

C.1. Forma de uso

Las etiquetas disponibles en el motor de plantillas para Virtual Museum conforman un conjunto de funcionalidades en las que se puede apoyar el diseñador ya que generan automáticamente porciones de texto o código HTML útiles con las que componer una plantilla usable por varios museos virtuales.

Dentro de un archivo de plantilla la invocación a cada etiqueta presenta la siguiente sintaxis:

```
#etiqueta()
```

Como se ve, todas las etiquetas comienzan por '#' y terminan por paréntesis. Las etiquetas disponibles junto con su descripción se enumeran en el siguiente apartado.

Hay que hacer notar que excepcionalmente hay etiquetas que necesitan un parámetro adicional en el que se indica alguna propiedad, como es el caso de la etiqueta *menu*. Para ellas, la sintaxis necesaria varía ligeramente para incluir el valor del parámetro:

```
#menu(<nombreMenu>)
```

En esta etiqueta se añade dentro de los paréntesis el parámetro de *menu*, que en este caso se corresponde con el nombre del menú que se desea mostrar.

C.2. Referencia de etiquetas

selectorIdioma

Produce el conjunto de enlaces para cambiar el idioma de los contenidos a partir de los configurados en el museo virtual.

nombreMuseo

Muestra el nombre del museo.

plantilla

Devuelve la ruta de la plantilla para que en base a ella se pueda referenciar a las hojas de estilo CSS, imágenes, javascript, etc que van incluidos dentro de la plantilla. De esta forma, si el diseñador por ejemplo ha incorporado una hoja de estilos llamada estilosGenerales.css en el paquete de su plantilla dentro de un directorio llamado 'css', dentro de los archivos de plantilla podría hacerle referencia mediante:

```
<link href="#plantilla()/css/estilosGenerales.css"
                    media="screen" rel="stylesheet" type="text/css" />
```

tema

Alias de *plantilla*.

publicidad

Muestra el contenido del código HTML configurado en el apartado de publicidad dentro del museo virtual.

rss

Genera un enlace que referencia al feed RSS con los contenidos actualizados dentro del último mes.

jarMovil

Genera los enlaces necesarios para referenciar a los recorridos para el móvil generados. Esta etiqueta brinda la posibilidad de enlazar los recorridos para el móvil dentro del diseño general de la

plantilla. Además de ello, el usuario responsable del museo también podría descargarse el recorrido generado e importarlo a continuación como un fichero más gracias al Gestor Multimedia.

comentarios

Lista todos los comentarios dejados por los usuarios visitantes del museo.

nuevoComentario

Genera un enlace a un formulario que permite dejar un nuevo comentario. El formulario al que se redirecciona puede ser especificado dentro de la propia plantilla por el diseñador o bien dejar que Virtual Museum provea uno por defecto. Si lo incluye el diseñador de la plantilla, se deberían añadir los ficheros con el formulario para dejar el comentario (*nuevoComentarioForm.vm*) y la pantalla de aceptación del mismo (*comentarioOkForm.vm*). Para más información puede consultar el apartado A.1.5.

dublinCoreHTML

Genera los registros de metadatos Dublin Core para el contenido que se esté mostrando actualmente en forma de etiquetas *meta* dentro de la cabecera de la página HTML, como se muestra en el siguiente fragmento de código:

```
<html>
<head>
...
<meta name="DC.title" scheme="DCTERMS.W3CDTF" content="Obra de Prueba" />
...
</head>
...
```

Si el contenido actual es una obra se podrán generar el conjunto completo de etiquetas Dublin Core, mientras que si no lo es, sólo se generarán las etiquetas *identificador, título, descripción e idioma*.

contenido

La etiqueta *contenido* se encarga de *renderizar* el contenido especificado en la URL, enmarcado dentro de la plantilla del museo.

categoria

La etiqueta *categoria* se encarga de *renderizar* los contenidos especificado para la categoría dada en la URL, enmarcado dentro de la plantilla del museo.

categoriaItem

La etiqueta *categoriaItem* se encarga de *renderizar* el contenido con el texto ampliado para la categoría especificada en la URL y el item seleccionado, enmarcado dentro de la plantilla del museo.

includeCSS

Dentro del conjunto de archivos incluidos desde el Gestor Multimedia el usuario responsable del museo puede añadir hojas de estilo CSS con las que refinar el diseño de su propio museo virtual. Para que estos estilos CSS estén disponibles dentro del sitio web completo, se ha de incluir esta etiqueta dentro de la cabecera (en el *head* del html) de los archivos de plantillas.

Por cada archivo de hojas de estilo CSS que se haya incluido el motor generará una línea de inclusión como la siguiente:

```
<link href="<ficheroCSS>" media="screen" rel="stylesheet" type="text/css" />
```

includeEffects

Añade los scripts Javascript necesarios para permitir algunos efectos de visualización, como por ejemplo la navegación por las imágenes de un contenido o categoría.

menu

La etiqueta *menu* presenta un parámetro adicional que indica el nombre del menú a *renderizar*. Su sintaxis es la siguiente:

```
#menu("menu1")
```

done menu1 se debe corresponder con algún nombre de menú dentro del propio museo virtual. Por eso, los menús permiten tener un nombre propio y otro nombre para plantilla. El primero es el usado por el responsable del museo, mientras que el segundo se ha de hacer corresponder con el que viene dado por la propia plantilla, por lo que sería recomendable que el diseñador haga notar en las instrucciones de la plantilla qué nombres ha elegido para los menús que incorpora su plantilla.

Especificando el nombre el menú a mostrar, esta etiqueta genera la lista de items del menú especificado en forma lista desordenada o, dicho de otro modo, etiquetas <*ul*> y <*li*>. A menudo los menús desplegables suponen una fuente de problemas al usuario, tanto de accesibilidad como de usabilidad. Es por ello que la alternativa escogida para su generación se basa en etiquetas <*ul*>, ya que combina la accesibilidad con la posibilidad de personalización mediante CSS o incluso mediante JavaScript, entre otras técnicas.

enlaceVolverCategoria

Si la opción elegida para *renderizar* una categoría de contenidos es mostrar un texto introductorio más un enlace referente al texto completo, dentro del archivo de plantilla del item de categoría puede usarse esta etiqueta para volver a visualizar la categoría completa, ya que genera un enlace que apunta a la categoría desde la que se venía. El diseñador puede hacer uso de esta etiqueta y se recomienda su empleo antes que la utilización de JavaScript a través del objeto *history*, ya que podría dar efectos indeseados debido a la inclusión de otros scripts dentro de la página web completa.

Apéndice D

CONTENIDO DEL CD

D.1. Contenidos

En este apéndice se enumeran los distintos contenidos incluidos en el CD entregado junto con la memoria impresa.

D.1.1. Código Fuente

/museum4j

Los archivos fuente del sistema están dividido según los bloques que lo conforman. A grandes rasgos se pueden mencionar los siguientes directorios:

- **/WEB-INF:** Archivos XML de configuración de la aplicación para los componentes de Struts, Tiles, Spring, Hibernate, Castor, Ant, Velocity y Log4j, y de la propia aplicación instalada en el contenedor (*web.xml*)
- **/WEB-INF/src:** Conjunto de archivos de código fuente organizados según la jerarquía de paquetes y clases expuesta en la fase de diseño.
- **/imagenes**, **/include** y **/recursos:** Imágenes, CSS y Javascript.
- **/siria:** Archivos JSP para la vista de la aplicación del responsable del museo.
- **/hestia:** Archivos JSP para la vista de la aplicación del administrador.
- **/geminis:** Archivos Velocity para la vista de la aplicación del visitante.
- **/hermes:** Archivos base y temporales para la generación del *jar* de la aplicación para dispositivos móviles.
- **/httpError:** Archivos JSP de errores personalizados del servidor (400, 404 y 500).

D.1.2. Scripts de Instalación

/virtualmuseum

Está compuesto por los distintos *shellscripts* y archivos adjuntos necesarios para la instalación y puesta en marcha del sistema.

- **/backup:** Script para hacer copias de seguridad de la base de datos MySQL
- **/install:**
  - **/bin:** Script principal y auxiliares para la instalación (*instalar.sh*)
  - **/logs:** Archivos de *log* que muestran información adicional de seguimiento para cada instalación efectuada en la máquina.
  - **/museum4j.war:** Aplicación Virtual Museum lista para ser desplegada en el contenedor web Tomcat.
  - **/scripts:** Scripts auxiliares a la aplicación.
  - **/sql:** Script SQL de creacion de tablas y poblado inicial usado durante la instalación para la inicialización de la aplicación.
- **/restore:** Script para reinicializar la base de datos del sistema.

D.1.3. Memoria

/documentos

La presente memoria está escrita íntegramente en LaTeX por lo que los fuentes que generan el archivo final **memoria.pdf** (situado en la raíz del CD) se basa en ficheros *.tex*. El texto completo se ha divido en un fichero por cada capítulo o índice, más uno maestro que define la estructura general del documento resultante [1]:

- **/memoria.tex:** Documento maestro de la memoria.
- **/capitulos:** Fuentes para los capítulos y apéndices; los capítulos están divididos según las materias que cubren: introducción, panóramica de museos, Dublin Core, planificación, análisis, diseño e implementación. conclusiones.
- **/capitulos/memoria.bib:** Archivo de referencias bibliográficas incluido en el documento maestro.
- **/portada:** Fuente para la primera página que contiene la portada, situada fuera de */capitulos* al tratarse de una página especial en cuanto a diseño.

[1] A partir del documento maestro se podría obtener el pdf (o ps, dvi, html, etc) de cada capítulo individualmente con tal de fijar su inclusión o no, de tal forma que resulte en un documento muy configurable

- **/estilo:** Ficheros de comandos TEXpara la inclusión de citas (*citas.tex*) y formateo de títulos de capítulo y apéndices (*formatoTitulos*).
- **/imagenes:** Imágenes empleadas a lo largo de los capítulos; están divididas según las materias que cubren, de la misma forma que para división de */capitulos*.

D.1.4. Anexos

/anexos

Este directorio reúne diversa documentación no recogida en la memoria impresa que resultará útil para ampliar la ciertas secciones de la misma:

- **/together:** Proyectos de Together Designer y Together Arquitect del análisis y diseño del proyecto.
- **/php:** Aplicación web escrita en PHP como prueba de concepto para el uso del servicio web expuesto por Virtual Museum.
- **/virtualmuseum.es:** Sitio web para la exploración y publicación de los archivos KML de GoogleMaps, y del mapa del sitio (*sitemap*) asociado, destinado a la indexación por parte de los motores de búsqueda de Internet.

Apéndice E

PUBLICACIÓN DE MAPAS

E.1. Introducción

La publicación de mapas en Google no es un proceso a ciencia cierta, aunque se puede facilitar el camino para que se indexen los mapas que genera Virtual Museum. Para enviar geocontenidos a Google, uno de los procesos a seguir se basa en el empleo de *sitemaps* y archivos KML de mapas, cuyos fundamenteos veremos a continuación.

E.2. Sitemaps

Los *sitemaps* pueden mejorar la optimización para los motores de búsqueda de un sitio asegurándose que todas ellas puedan ser encontradas. La mayoría de los motores de búsquedas solo seguirán un finito número de enlaces desde una página, así si el sitio es muy grande, se deberán usar estrategias adicionales además del *sitemap* requerido por los motores de búsqueda y para que los visitantes puedan acceder al contenido.

E.3. Especificación del formato XML

El protocolo Sitemap se forma a partir de etiquetas XML, donde todos los valores deben estar escapados (*entity-escaped*).

El sitemap debe cumplir las siguientes características:

- Tener el nodo raíz con la etiqueta <*urlset*>
- Especificar el espacio de nombres dentro de esta etiqueta
- Incluir una entrada <*url*> por cada dirección URL
- Incluir una entrada <*loc*> dentro de cada una de las entradas <url>anteriores

El resto de etiquetas son opcionales y su soporte puede variar entre los distintos motores de búsqueda. Las etiquetas disponibles se describen a continuación:

- **<*url*>:** Etiqueta padre de cada entrada de URL.
- **<*loc*>:** URL de la página: debe empezar por el protocolo (http por ejemplo) y terminar por barra (/), si el servidor web lo requiere. Este valor debe presentar una longitud menor de 2048 caracteres.
- **<*lastmod*>:** La fecha de la última modificación del fichero, en formato de fechas W3C. Se puede omitir la porción de la hora, y usar entonces YYYY-MM-DD. Nótese que esta etiqueta es independiente de la cabecera 304 (if-modified-since) que puede devolver un servidor web, y los motores de búsqueda pueden usar la información de ambas fuentes de maneras distintas.
- **<*changefreq*>:** Frecuencia con la que la página espera cambiarse. Este valor da información general a los motores aunque no se corresponde con la frecuencia con que éstos barrerán la pagina. Los valores válidos son: *always, hourly, daily, weekly, monthly, yearly, never*. El valor *always* debería usarse para describir documentos que cambian cada vez que son accedidos, mientras que el valor *never* debería emplearse para referenciar URLs archivadas.
- **<*priority*>:** Prioridad relativa de la URL respecto del sitio entero.

E.4. Google Maps

Dentro de los archivos sitemap se pueden albergar datos para Google con información geográfica haciendo referencia a archivos KML dentro de una etiqueta <*url*>.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Las estructura de este fichero se descompone de la siguiente manera:

- Cabecera XML: es la primera línea del archivo; no se permiten espacios y otros caracteres antes de esta línea
- Declaración el espacio de nombres KML
- Objeto *Placemark* que contiene los elementos:
  - *name*, nombre usado como la etiqueta para el Placemark
  - *description*, descripción que aparece en el ”globo” adjunto al Placemark

    - *point* , especifica la posición del placemark en la superficie terrestre (longitud, latitud y de forma opcional la altitud)

- Placemark: nombre, descripción, y coordenadas del punto

Bibliografía

[ADe08] ADeveria. Timeline of web browsers. *en.wikipedia.org*, Última consulta: agosto 2008.

[Amb02] Scott Ambler. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. Wiley Publishing Inc., 2002.

[Amb08] Scott Ambler. Agile modeling. *www.agilemodeling.com*, Última consulta: agosto 2008.

[Arm00] William Y. Arms. *Digital Libraries*. The MIT Press, 2000.

[Ash04] Ashmore, Derek. *The J2EE architect's handbook*. DVT Press, 2004.

[BHP02] Andy Longshaw Bond, Martin Haywood, Dan, Debbie Law and Peter Roxburgh. *Sams Teach Yourself J2EE in 21 Days*. Sams Publishing, 2002.

[BK05] Bauer, Christian and King, Gavin. *Hibernate in Action*. Manning Publications Co., 2005.

[Bra06] Alejandro Bravo. Webposible. *www.webposible.com*, Última consulta: octubre 2006.

[Cor07] Dublin Core. Dublin core metadata initiative (dcmi). *Dublin Core site www.dublincore.org*, Última consulta: julio 2007.

[DA97] Duguid, Paul and Atkins, Daniel E. *Report of the Santa Fe Planning Workshop on Distributed Knowledge Work Environments: Digital Libraries*. University of Michigan School of Information, 1997.

[DT03] De Carli, Georgina and Tsagaraki, Christina. Los museos latinoamericanos e internet: la experiencia de la red-ilam. *Instituto Latinoamericano de Museos Costa Rica*, Agosto 2003.

[eS07] Stamelos, Ioannis et Sfetsos, Panagiotis. *Agile Software Development Quality Assurance*. Information Science Reference, 2007.

[Est05] Assumpció Estivill. Uso de metadatos dublin core en la descripción yrecuperación de artículos de revistas digitales. *Proceedings DC-2005 International Conference on Dublin Core and Metadata Applications*, 10 2005.

[For04] Ford, Neal. *Art of Java Web Development*. Manning Publications Co., 2004.

[GC03] Gradecki, Joseph D. and Cole, Jim. *Mastering Apache Velocity*. Wiley Publishing Inc., 2003.

[Gue08] Antonio Guerrero. Las bibliotecas en el entorno digital. *www.documentaciondigital.net*, Última consulta: marzo 2008.

[Hol05] Steven Holzner. *Java after hours: 10 Projects you'll never do at work*. Sams Publishing, 2005.

[Huh02] Erkki Huhtamo. On the origins of the virtual museum. *Nobel Symposium (NS 120) - Virtual Museums and Public Understanding of Science and Culture*, Mayo 2002.

[Ive04] Will Iverson. *Hibernate: A J2EE Developer's Guide*. Addison Wesley Professional, 2004.

[JH04] Johnson, Rod and Hoeller, Juergen. *J2EE Development without EJB*. Wiley Publishing Inc., 2004.

[Joh02] Johnson, Rod. *J2EE Design and development*. Wiley Publishing Inc., 2002.

[JR00a] Booch, Grady Jacobson, Ivar and Rumbaugh, James. *El Lenguaje Unificado de Modelado. Manual de Referencia*. Addison Wesley, 2000.

[JR00b] Booch, Grady Jacobson, Ivar and Rumbaugh, James. *El Proceso Unificado de desarrollo software*. Addison Wesley, 2000.

[Keo02] Jim Keogh. *J2EE, Manual de referencia*. Mc Graw Hill, 2002.

[Kur03] Martin Kurth. Repurposing marc metadata: using digital project experience to develop a metadata management design. *Library Hi Tech, Volume 22, Nº22*, 11 2003.

[LS03] Letelier Torres, Patricio and Sánchez López, Emilio. Metodologías Ágiles en el desarrollo de software. *VIII Jornadas de Ingeniería del Software y Bases de Datos. Grupo ISSI. Universidad Politécnica de Valencia*, Noviembre 2003.

[M+00] Maruyama, Hiroshi et al. *Creación de sitios web con XML y Java*. Prentice Hall, 2000.

[M+03] McGovern, James et al. *Java 2 Enterprise Edition 1.4 Bible*. Wiley Publishing Inc., 2003.

[MP02] Molpeceres, Alberto and Pérez, Martín. Arquitectura empresarial y software libre. *www.javahispano.org*, Agosto 2002.

[MU08] Jose Ángel Martínez Usero. Blog sobre interoperabilidad y asuntos relacionados. *interoperabilidad.blogspot.com*, Última consulta: junio 2008.

[Mén04] Eva María Méndez. La web semántica: una web más bibliotecaria. *Boletín de la SEDIC CLIP n.41*, 2004.

[Pas04] Thomas B. Passin. *Explorer's guide to the Semantic Web*. Manning Publications Co., 2004.

[Pér03] Ángel Pérez. Aplicación de generación de visitas virtuales para pequeños y medianos museos de arte. *Proyecto fin de carrera Ingeniero en Informática. Universidad de Valladolid*, Septiembre 2003.

[RF04] Robinson, Mike and Finkelstein, Ellen. *Jakarta Struts For Dummies*. Wiley Publishing Inc., 2004.

[Ric06] Richardson, Chris. *POJOs in Action*. Manning Publications Co., 2006.

[RV07] Laura Regil Vargas. Museos virtuales: nuevos balcones digitales. *(www.mexicandesign.com) Universidad Autónoma Metropolitana - Distrito Federal, México*, Última consulta: agosto 2007.

[WB05] Walls, Craig and Breidenbach, Ryan. *Spring in Action*. Manning Publications Co., 2005.

[Wei04] Weitzenfeld, Alftedo. *Ingeniería del Software orientada a objetos con UML, Java e Internet*. Thomson, 2004.

[ZS06] F.J. Zarazaga-Soria. El papel de dublin core en el desarrollo de las infraestructuras de datos espaciales. *Avances en las infraestructuras de datos espaciales*, Abril 2006.