Esta tesis doctoral contiene un índice que enlaza a cada uno de los capítulos de la misma.
Existen asimismo botones de retorno al índice al principio y final de cada uno de los capítulos.

Ir directamente al índice

Para una correcta visualización del texto es necesaria la versión de Adobe Acrobat Reader 7.0 o posteriores

Aquesta tesi doctoral conté un índex que enllaça a cadascun dels capítols. Existeixen així mateix botons de retorn a l'index al principi i final de cadascun deis capítols .

Anar directament a l'index

Per a una correcta visualització del text és necessària la versió d' Adobe Acrobat Reader 7.0 o posteriors.

Universitat d'Alacant
Universidad de Alicante

OO-H: UNA EXTENSIÓN A LOS MÉTODOS OO PARA EL MODELADO Y GENERACIÓN AUTOMÁTICA DE INTERFACES HIPERMEDIALES

Memoria de Tesis Doctoral

Autor: **Cristina Cachero Castro**

Directores:
Dr. Jaime Gómez Ortega
Dr. Óscar Pastor López

Depto. de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Alicante, 2 de diciembre de 2002

El tribunal encargado de juzgar esta tesis doctoral está constituido por los siguientes doctores:

Presidente:

Dr. Isidro Ramos (Universidad Politécnica de Valencia)

Secretario:

Dr. Manuel Palomar (Universidad de Alicante)

Vocales:

Dr. Piero Fraternali (Politécnico de Milán)

Dra. Nora Koch (Universidad de Munich)

Dr. Mario Piatini (Universidad de Castilla la Mancha)

Esta tesis ha sido parcialmente patrocinada por la Conselleria de Cultura, Educació i Ciència de la Comunitat Valenciana y por el Ministerio de Ciencia y Tecnología (proyecto TIC2001-3530-C02-02)

Resumen de la Tesis

En la actualidad, los métodos de modelado conceptual de aplicaciones centran la mayor parte de sus esfuerzos en el correcto modelado de sus capas estática, dinámica y funcional, pero suelen prestar poca o ninguna atención a la presentación, es decir, a su interfaz. Sin embargo, el diseño cuidadoso de esta interfaz, así como sus características de facilidad de uso, evolución y mantenimiento, son aspectos fundamentales para el éxito final de la aplicación, y determinan el nivel de satisfacción alcanzado por el usuario durante el uso de la misma.

Esta carencia se ha visto acentuada con la irrupción del *World Wide Web* (WWW) como entorno de desarrollo de aplicaciones. Las interfaces en web son en general mucho más volátiles que las interfaces tradicionales, y sus usuarios suelen mostrar menor tolerancia ante información o enlaces erróneos. Las herramientas existentes, diseñadas con la noción, ya obsoleta, de web como mero medio difusor de información, se han mostrado inadecuadas para cubrir todo el proceso de producción de este tipo de aplicaciones software de forma unificada y sistemática, desde la especificación precisa de requisitos hasta su implementación final. Así, la construcción y puesta en funcionamiento de sitios web atractivos y complejos, con funcionalidad no trivial, requiere nuevas aproximaciones que faciliten el trabajo del desarrollador web de manera consistente y eficiente. Este proceso de diseño sistemático es aún más importante si tenemos en cuenta que la web es un entorno en continua evolución, donde nuevas tecnologías, cada vez más potentes, están constantemente emergiendo. Además, la existencia de modelos conceptuales, independientes del lenguaje final de implementación, agiliza la adaptación de las aplicaciones web a

dichos cambios de entorno en un período de tiempo razonable, al favorecer la reutilización de la información capturada durante el proceso de análisis y diseño.

Como propuesta para cubrir esta carencia, en esta tesis se presenta un método conocido como *Object-Oriented Hypermedia Method* (OO-H) que, utilizando una aproximación Orientada al Objeto, captura la semántica necesaria para el modelado eficiente de interfaces de usuario y su implantación en web, y extiende por tanto los métodos existentes de modelado conceptual de aplicaciones.

OO-H recoge las dimensiones de navegación y presentación características de este tipo de interfaces mediante tres tipos de diagrama: el Diagrama de Acceso Navegacional (DAN), el Diagrama de Presentación Abstracta (DPA) y el Diagrama de Diseño Visual (DDV). El DAN toma como base los requisitos funcionales de cada tipo de usuario y el modelo conceptual de dominio del sistema para, a partir de ellos, construir un modelo que refleje los caminos navegacionales que puede recorrer el usuario a través del espacio de información para dar respuesta a los requisitos identificados. Para ello, el DAN proporciona los constructores necesarios para especificar aspectos como el modo de acceso a la información o el modo de invocación de servicios. A partir del DAN es posible generar una estructura de plantillas XML que describen de una manera textual la interfaz modelada. De entre ellas, el DPA representa de una manera diagramática aquéllas correspondientes a documentos de datos (páginas abstractas) y enlaces entre dichos documentos de datos. Estas plantillas XML pueden ser ulteriormente refinadas por el diseñador para conseguir los rasgos de presentación deseados a través de distintos DDV, uno por cada página abstracta contenida en el DPA.

Para refinar los distintos diagramas de un modo sistemático, OO-H integra un Lenguaje de Patrones que se almacena en un Catálogo de Patrones. Este lenguaje de patrones captura reglas de diseño que ayudan a incrementar la calidad y facilidad de uso de la interfaz.

Como resultado de todo este proceso, una interfaz de aplicación web integrable con módulos de lógica preexistentes puede ser generada de forma automática a partir de esta especificación.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todos los que, a nivel profesional y, sobre todo, a nivel personal, han hecho posible que este trabajo haya sido finalizado.

A los codirectores de esta tesis, el Dr. Jaime Gómez (Universidad Alicante) y el Dr. Oscar Pastor (Universidad Politécnica Valencia), por su ayuda y soporte a lo largo del presente trabajo. Muchas buenas ideas han partido de nuestras conversaciones, y han sido un apoyo científico constante durante el desarrollo de la misma. A Antonio Párraga, el programador más efectivo que conozco. Gran parte de lo que es OO-H hoy en día ha sido posible gracias a su increíble trabajo en Visual WADE, la herramienta CAWE que le da soporte.

A mis compañeros de trabajo, en especial a Irene Garrigós, Rafael Muñoz, Juan Carlos Trujillo, Rafael Romero, José Luis Vicedo y Andrés Montoyo. Por hacer de paraguas en muchas ocasiones, y por todo su ánimo y apoyo en estos últimos meses. A José Ramón Lillo, nuestro incansable administrativo (que no secretario :) por estar siempre ahí, dispuesto a prestarme su hombro, amigo además de compañero. A Alicia Garrido, Enrique Medina y a mis compañeros de turismo, por todos los reajustes de distribución de trabajo y horarios que han realizado para que yo pudiese terminar esta tesis. A Marcela Genero, que siempre me hace ver la parte positiva de las situaciones. A Nora Koch, por su trabajo, por sus siempre acertados comentarios y sobre todo por transmitirme su gusto por el trabajo bien hecho. A Carlos Cachero, mi único hermano :). A Lourdes Tapiador, a Manoli Maciá, a Santiago Meliá, a Concha Toribio, a Fernando Fuentes, a Juan

José Haro y a Ana Guijarro, por todos los cafés y todos los planes pospuestos 'hasta que Cristina acabe la tesis'... chicos, ¡id preparando las maletas! :) Y a Antonio Hernández... por su sentido del humor, por su estoico aguante ante mis ataques de pánico y por todo su cariño.

Sobre todo, quisiera dar las gracias a mis padres, Antonio Cachero y Amalia Castro... por su amor y apoyo incondicional. Sin vosotros nada de esto habría sido posible.

Alicante, Octubre 2002 *Cristina Cachero*

Índice General

Índice de Figuras

1. Aplicaciones web: un nuevo reto para la Ingeniería del Software

No puedo trabajar sin un modelo. No diré que no cierro implacablemente los ojos a la naturaleza con el fin de convertir un estudio en un cuadro, organizando los colores, alargando y simplificando; pero en cuestiones de forma temo demasiado alejarme de lo posible y lo verdadero.

VINCENT VAN GOGH

1.1 Motivación

En los últimos años hemos sido testigos de cómo los avances en Internet en general y la World Wide Web (WWW) en particular[1] han creado un entorno de desarrollo distribuido mucho más flexible, donde se han materializado nuevas oportunidades de negocio (e.g. aplicaciones de venta por internet[2] o sistemas de subastas en línea[3]). Algunas de las características de la web responsables de su rápida implantación son su ubicuidad, la existencia de estándares abiertos, el enlazado de información, la facilidad de acceso a datos y servicios o la facilidad de creación de contenidos. Estos rasgos han impulsado en el seno de las empresas no sólo la inversión en nuevos tipos de Sistemas de Información (SI), sino también la migración de aplicaciones empresariales tradicionales a este nuevo entorno, con la expectativa de disminuir los presu-

[1] Partiendo de la definición de Internet como 'red de redes', la WWW puede ser vista como el subconjunto dentro de Internet más dinámico, en términos tanto de crecimiento como de avances técnicos de los que se ha beneficiado

[2] B2C e-commerce applications

[3] auction systems (considerados por algunos autores como un tipo particular de aplicación de comercio electrónico C2C)

puestos de desarrollo y mantenimiento e inducir una menor curva de aprendizaje para nuevos usuarios del sistema.

Sin embargo, estas expectativas no siempre se están viendo reflejadas en la práctica. El motivo es que, aunque actualmente los SI son cada vez más complejos y dinámicos en términos de estructura, contenido, comportamiento e interfaz, la implantación de aplicaciones desarrolladas en la web (a partir de ahora *aplicaciones web*) se sigue realizando en su mayor parte con un proceso de desarrollo creativo, poco estructurado, que tiene sus orígenes en la premisa, ya obsoleta, de aplicación web como *kiosco de información*[4], i.e. restringida al ámbito de la navegación hipertextual de documentos. Este hecho está generando grandes dificultades de uso, funcionalidad, evolución y mantenimiento de las aplicaciones implantadas, hasta tal punto que ya se ha hablado de la amenaza de una *Crisis del web* (Murugesan *et al.*, 1999; Ginige & Murugesan, 2001), para la que se aducen razones similares a las esgrimidas para justificar la crisis del software de los años 80.

La necesidad de evitar dicha crisis y manejar los riesgos inherentes a la creciente complejidad de las aplicaciones demandadas por las organizaciones para su explotación en Internet ha sido el principal detonante de la aparición, en los últimos años, de un amplio rango de métodos, técnicas y procesos ingenieriles que, englobados bajo el epígrafe común de *Ingeniería Web*[5], se orientan a facilitar la comprensión, desarrollo, evolución y mantenimiento de las aplicaciones web. Estas soluciones han supuesto en la mayoría de los casos la extensión de técnicas aplicadas en la ingeniería del software tradicional con nuevos constructores y vistas hipermediales que abordan el problema de la navegación del usuario a través del espacio de información. Sin embargo, la idiosincrasia de la web introduce una serie de nuevos retos que van mucho más allá del planteamiento de este mapa de navegación, y que incluyen

[4] *web kiosks*

[5] Web Engineering (WebE). Se trata de una nueva disciplina dentro de la Ingeniería del Software, cuyo objetivo es englobar y unificar toda una serie de prácticas ingenieriles adaptadas para hacer frente a los retos específicos que presentan las aplicaciones web (Ginige & Murugesan, 2001). Para una definición más exhaustiva los lectores son remitidos a (WebE, 2002).

aspectos como necesidad de evolución continua, margen de error cero, disponibilidad global de fuentes de información o tratamiento de una audiencia heterogénea y, a menudo, imposible de definir *a priori* (Fraternali & Paolini, 2000; Lowe & Henderson-Sellers, 2001; R.S. Pressman, 2000).

Desde este trabajo pensamos que es fundamental que las tareas para los informáticos que trabajan en las áreas relacionadas con los SI en web se enfoquen hacia el desarrollo y evolución de procesos, métodos, herramientas y técnicas que manejen adecuadamente las peculiaridades de esta nueva plataforma, y ayuden a dar respuesta a las principales metas de los desarrolladores: mayor productividad, mayor velocidad de desarrollo y menor coste de mantenimiento de las aplicaciones.

En el resto del capítulo nos centraremos en definir qué debemos entender cuando, en el contexto del presente trabajo, hablamos de *aplicación web*, cuáles son sus características más relevantes y qué retos concretos plantea a la comunidad científica su desarrollo e implantación.

1.2 Concepto de aplicación web

Definimos aplicación web[6] como un SI donde una gran cantidad de datos volátiles, altamente estructurados, son consultados, procesados y actualizados mediante navegadores. Precisamente una de las características principales de las aplicaciones web tal y como se entienden en el presente trabajo es su alto grado de interacción con el usuario. Por otro lado, la complejidad de la funcionalidad ofertada por dichas aplicaciones puede variar desde la más básica (navegación/consulta, altas, bajas, modificaciones[7]) hasta complejos servicios transaccionales como los encontrados tras las aplicaciones de comercio electrónico. Otro rasgo característico de este

[6] Algunos autores hablan de SI basados en web (*Web-based Information Systems o WIS*). En el contexto de este trabajo, este término se considera sinónimo al de aplicación web.

[7] Estas operaciones son a menudo referidas como operaciones CRUD, acrónimo proveniente de las palabras inglesas *Create, Read, Update, Delete.*

tipo de aplicaciones es que el diseño de interfaz está condicionado por las necesidades de claridad y simplicidad más que por un diseño impactante. Además, dicha interfaz debe estar estructurada de manera que oriente a cada tipo de usuario en función de sus necesidades particulares de información/funcionalidad a través del el espacio de información.

Ateniéndonos a esta definición, es posible realizar una clasificación de los distintos tipos de aplicaciones web que encontramos en la actualidad.

1.2.1 Tipos de aplicaciones web

Desde el punto de vista de los desarrolladores de aplicaciones, la diferenciación entre distintos tipos de aplicaciones web puede atender a criterios tan dispares como la complejidad de datos y de la propia aplicación (Mecca *et al.*, 1999a), la volatilidad y estructuración de los datos (Isakowitz *et al.*, 1995) o la intencionalidad de la aplicación (Dart, 1999; Ginige & Murugesan, 2001; R.S. Pressman, 2000).

De entre ellos, resulta especialmente relevante el criterio de *intencionalidad de la aplicación*, que da lugar a la siguiente categorización[8]:

- Informacionales: orientadas a la diseminación de información, personalizada o no, y con acceso a base de datos o sin él. Dentro de esta categoría se engloban los periódicos en línea, catálogos de productos, etc.
- Orientadas a descarga de datos: servidores de materiales didácticos, servidores de canciones, etc.
- Interactivas: orientadas a la interacción con el usuario. Ejemplos de este tipo de aplicaciones son los formularios de registro, sistemas de encuestas, etc. Su mecanismo principal de comunicación suele ser el formulario más que la página de información.

[8] Se ha de tener en cuenta que una misma aplicación puede pertenecer a más de una categoría

- Orientadas al servicio: sistemas de ayuda financiera, simuladores etc.
- Transaccionales: compra electrónica, banca electrónica, etc.
- De flujo de datos: sistemas de planificación en línea, manejo de inventario, monitorización de estados, etc.
- Entornos de trabajo colaborativo: sistemas de autoría distribuidos, herramientas de diseño colaborativo, etc.
- Comunidades on-line, sistemas C2C[9]: foros de debate, sistemas recomendadores de productos o servicios, sistemas de subastas, mercados en línea, etc.
- Portales web: intermediarios en línea, centros comerciales de compra electrónica, etc.
- Orientadas al análisis de datos: aplicaciones de almacenes de datos[10], aplicaciones OLAP[11], etc.

Esta categorización implica dejar fuera del concepto de aplicación web (y por tanto fuera del alcance de este trabajo) aplicaciones, también presentes en Internet, como los juegos en línea o las aplicaciones centradas en aspectos puramente multimediales.

Una vez delimitado este concepto, a continuación pasamos a detallar las características que diferencian a este tipo de aplicaciones de otros sistemas software.

1.2.2 Características y requisitos de las aplicaciones web

Las aplicaciones web, tal y como las entendemos en este trabajo, tienen una serie de rasgos comunes que, desde distintas perspectivas, la diferencian de otros tipos de aplicaciones software, y que son:

[9] *Customer to Customer*
[10] datawarehousing
[11] On-line Analytical Processing

- Desde el punto de vista del **usuario**, se ha universalizado su accesibilidad: actualmente un usuario experto y un usuario con habilidad limitada en el uso de aplicaciones informáticas acceden al mismo tipo de aplicación. Aún más, el número y tipo de usuario de las aplicaciones web no siempre es predecible, lo que obliga a tener el concepto de facilidad de uso aún más presente que en otros tipos de aplicaciones.
- Desde el punto de vista de la **información**, asistimos en la actualidad a una disponibilidad global de fuentes heterogéneas de información, estructurada y no estructurada, pertenecientes a distintos dominios y que colaboran en el cumplimiento de los objetivos de la aplicación.
- Desde el punto de vista de la **plataforma** se realiza un uso intensivo de la red y la conexión se establece desde distintos tipos de dispositivo de acceso.

Cada una de estas perspectivas introduce una serie de requisitos que deben ser tenidos en cuenta durante el proceso de desarrollo de cualquier tipo de aplicación web con el fin de incrementar su probabilidad de éxito de implantación (Fraternali, 1999; R.S. Pressman, 2000; Lowe & Henderson-Sellers, 2001), y que pueden ser estructuradas como sigue:

- Requisitos de desarrollo
  - **Portabilidad**. Debido a la dinamicidad del entorno tecnológico, a menudo es necesario implantar una misma aplicación en distintas plataformas, con distintas arquitecturas, distintas tecnologías y/o atendiendo a distintos dispositivos de acceso, lo que obliga a desarrollar técnicas, modelos y herramientas que faciliten el reuso e independicen hasta donde sea posible el desarrollo de la aplicación de consideraciones finales de implementación.
  - **Inmediatez (Rapidez de Implantación)**. El desarrollo de aplicaciones web requiere un período de implantación mucho más reducido, que influye en todo su ciclo de desarrollo.

- **Uso de herramientas de comunicación.** La idiosincrasia de este tipo de aplicaciones requiere a menudo la colaboración de un equipo de desarrollo heterogéneo, con personal tanto técnico como no técnico (diseñadores gráficos, proveedores de información, expertos en marketing, etc.), lo que obliga a planificar los canales y disponer de mecanismos estándares de comunicación durante el proceso de desarrollo.
- **Creación de contenidos** como parte integrante de la fase de ingeniería de la aplicación. Aunque en este trabajo nos centramos en la especificación de aplicaciones orientadas a ofrecer funcionalidad compleja, más allá de la mera diseminación de información, el diseño y producción de textos, gráficos, vídeos etc. que conforman la estructura informacional de la aplicación es una tarea que debería ser realizada en paralelo al diseño de la propia aplicación.
- **Integración (disponibilidad global) de fuentes heterogéneas de información.** La posible necesidad de manejo integrado de contenido estructurado y no estructurado, almacenado en distintos formatos (bases de datos, sistemas de ficheros, dispositivos multimedia) y accesibles de forma distribuida mediante múltiples aplìcaciones es otro de los factores que condiciona el proceso de diseño de este tipo de aplicaciones.

• Requisitos de aplicación

- **Evolución orgánica (continua).** Esta característica, aunque implícita en otros tipos de aplicaciones software, se convierte en un aspecto fundamental en el ámbito de la web, donde tanto el contenido como los requisitos de las aplicaciones evolucionan a una velocidad vertiginosa. Este hecho es en parte debido a que los clientes de este tipo de aplicaciones suelen tener un conocimiento muy pobre de sus necesidades y de las posibilidades del sistema durante las fases de inicio de vida del proyecto, lo cual complica la efectividad de la fase de análisis de requisitos. Además, la posibilidad que proporcionan las aplicaciones web de realizar cambios y que éstos

sean accesibles de manera inmediata, sin necesidad de ningún tipo de actuación por parte del usuario, ha incrementado la capacidad de soporte por parte de las organizaciones a esta evolución continua, y ha cambiado la naturaleza y requisitos del proceso de mantenimiento, dotándolo de un papel mucho más relevante dentro del ciclo de vida de la aplicación.

- **Seguridad en la comunicación**. Debido a que las aplicaciones web se encuentran disponibles a través de una red, es difícil limitar el grupo de usuarios finales que pueden acceder a ella. Es por ello que se hacen necesarios mecanismos (en la propia aplicación y en la plataforma que la soporta) para proteger información sensible y proporcionar modos seguros de transmisión de datos.

- **Calidad (margen de error cero)**. La permisividad mostrada por los usuarios ante los errores en aplicaciones web (ya se refieran a robustez, facilidad de uso o rendimiento) es muy limitada: enlaces erróneos o información desactualizada provocan la pérdida de usuarios de la aplicación. Este factor se ve potenciado por el hecho de que nos encontramos en un entorno abierto, con un bajo grado de fidelidad. Es por ello que en el desarrollo de este tipo de aplicaciones es primordial disponer de mecanismos exhaustivos de control de calidad que minimicen las posibilidades de fracaso de la aplicación.

- **Velocidad**. El uso intensivo de la red provoca que la elección de protocolos de comunicación y el mantenimiento de una velocidad de acceso adecuada sean una parte clave de diseño de dichas aplicaciones.

- Importancia de la **interfaz**. Ya hemos comentado cómo la red ha propiciado el acceso a las aplicaciones a usuarios con perfiles muy distintos. Esta característica de acceso universal ha agudizado la necesidad de implementar interfaces de usuario más intuitivas, capaces de capturar la atención del usuario y facilitar el acceso a la información a aquéllos que poseen una habilidad limitada en el uso de aplicaciones informáticas. De este modo, en aplicaciones web aspectos como el diseño

gráfico, la calidad cognitiva o la facilidad de uso tienen tanta relevancia como el propio diseño técnico de la aplicación.

- Necesidad de **personalización**. Debido, por un lado, a la facilidad de migración del usuario a otras aplicaciones que ofrecen servicios similares y que son fácilmente accesibles a través de la red, y por otro a la audiencia heterogénea de este tipo de aplicaciones, la personalización [12] se ha convertido en un elemento significativo del diseño, y da valor añadido a un contenido que debe además ser accesible y estar actualizado.

A estos requisitos debemos añadirles además los planteados para aplicaciones de software tradicional (seguridad de la propia aplicación, escalabilidad, disponibilidad, interoperabilidad con sistemas propietarios, etc.) que, obviamente, también siguen siendo aplicables en este nuevo entorno.

Todo lo mencionado anteriormente, unido a la heterogeneidad de la web (amalgama de tecnologías y dispositivos de acceso), obliga a integrar técnicas provenientes de campos tan diversos desde el punto de vista tecnológico como (R.S. Pressman, 2000) (1) el desarrollo de aplicaciones distribuidas (donde destaca el Desarrollo Basado en Componentes[13]), (2) técnicas de seguridad y (3) estándares de Internet.

Por un lado, la rápida expansión de sistemas basados en web se ha visto respaldada por un avance paralelo de estándares como COM/DCOM, CORBA, Enterprise JavaBeans o, en los últimos tiempos, los *Servicios Web*, que, de forma muy simplificada, permiten la comunicación entre componentes y/o sistemas construidos por distintos desarrolladores y ejecutados sobre distintas plataformas. El reuso de estas técnicas, y su integración en el proceso de desarrollo de interfaces hipermediales, agilizan de manera dramática el coste de desarrollo de nuevas aplicaciones web.

[12] En este trabajo definimos personalización como la capacidad que posee una aplicación de adaptarse al perfil de cada usuario con el fin de darle la impresión de que está trabajando con una aplicación específicamente diseñada para dar respuesta a sus necesidades particulares. Este concepto será desarrollado en más profundidad en el capítulo 7

[13] Component Based Development (CBD)

Por otro lado, como ya hemos comentado, el hecho de que una aplicación web resida en una red provoca que el acceso por parte de usuarios no autorizados sea mucho más factible. Para evitarlo existe un conjunto de medidas proporcionadas por la infraestructura de red (técnicas de encriptación, cortafuegos, etc.) que deben ser integradas en la aplicación.

Además, la existencia e idiosincrasia de protocolos y lenguajes estándares (HTTP, SOAP, HTML, XML, etc.), que se encuentran en la base del éxito de las aplicaciones web, deberían ser tenidos en cuenta durante el proceso de desarrollo de este tipo de aplicaciones.

Todo lo comentado hasta ahora sugiere la necesidad de definir un marco de desarrollo unificado, particular a este tipo de aplicaciones, que incluya un proceso general que tenga en cuenta de manera explícita las características particulares de las aplicaciones web, y que pasamos a detallar a continuación.

1.3 La Ingeniería del software orientada a la web

En la historia de los SI, la propuesta de técnicas de desarrollo y metodologías exhaustivas ha sido auspiciada a menudo por la aparición de nuevos tipos de aplicaciones, tal y como fueron en su momento los sistemas de apoyo a la decisión, los sistemas de trabajo colaborativo, etc. En estos nuevos tipos de aplicaciones, la falta de soporte inicial para el proceso y la carencia de prácticas de desarrollo adaptadas a su idiosincrasia produjo sistemas poco fiables, baja productividad y altos costes de mantenimiento, lo cual motivó el surgimiento de distintas corrientes de investigación para dar respuesta a los problemas detectados.

Éste ha sido también el caso de las aplicaciones web y, debido a que todavía nos encontramos en las primeras fases de desarrollo de este tipo de sistemas, no nos deben extrañar los problemas a

los que se enfrentan en la actualidad algunas aplicaciones web en explotación, como son (Ginige & Murugesan, 2001):

- Alta dependencia para su desarrollo e implantación de habilidades individuales y no de prácticas estándares
- Falta de facilidad de uso (navegación, accesibilidad)
- Falta de escalabilidad
- Dificultad de mantenimiento
- Dificultad de integración con otros sistemas
- Baja fiabilidad y seguridad

La solución a tales problemas también se presenta, como ocurrió en el pasado con otros tipos de aplicaciones, en forma de aplicación de técnicas ingenieriles al desarrollo de estos nuevos tipos de sistemas, entre las que se incluye el uso de una metodología coherente, un proceso disciplinado y repetible, mejores herramientas de desarrollo y un conjunto de guías de diseño eficaces. El uso de estas técnicas además debe producirse de tal forma que permita estructurar, comunicar, entender, simplificar y formalizar tanto el dominio del problema como las decisiones de diseño, así como disponer de una documentación detallada y exacta ante futuras modificaciones.

Este es precisamente el objetivo con el que nace la Ingeniería del Web: controlar el caos que han provocado en el pasado procesos creativos de desarrollo con el fin de proporcionar un proceso sistemático orientado a la mejora de la calidad de la aplicación final. En esta nueva disciplina se parte de la base de que las necesidades de evolución, mantenimiento, la adaptación a nuevos dispositivos de acceso y la migración a nuevas plataformas y entornos de desarrollo deben dirigir el proceso del ciclo de vida, hasta tal punto que algunos autores (Ginige, 1998) definen la Ingeniería Web como una disciplina de *diseño del cambio*.

1.3.1 Conceptos generales: metodología, método, proceso y modelo

Antes de entrar de lleno en la presentación de los distintos aspectos que intervienen en una definición ingenieril de las aplicaciones web, creemos que es importante clarificar qué debemos entender cuando en el presente trabajo hablamos de metodología, método, proceso y modelo, para lo cual hemos adoptado la definición aportada por (Evitts, 2000) que presentamos a continuación.

El término *metodología* se utiliza en este trabajo para definir un paquete formal de actividades, técnicas y productos finales, que cubre todo el ciclo de desarrollo de la aplicación.

Por *método* entendemos una versión simplificada y menos exhaustiva de una metodología. Los métodos se enfocan hacia las técnicas y elementos de ingeniería (e.g. modelos del producto software) que intervienen en un subconjunto de actividades de la metodología. Tanto las metodologías como los métodos suelen tener un lenguaje de modelado asociado.

Por su parte, un *proceso* define una estrategia apropiada para abstraer, organizar, ejecutar y/o controlar las distintas fases, tareas, recursos y artefactos de un proyecto con el objeto de alcanzar las metas establecidas. El proceso, como elemento integrante de una metodología, se centra en la secuenciación de actividades, tanto aquéllas en las que intervienen las técnicas definidas en la metodología como aquéllas que influyen en su implantación práctica (e.g. actividades de dirección, medición e implantación de la aplicación). El mismo concepto en el ámbito de un método se simplifica, y suele limitarse a secuenciar las actividades en las que aparecen las técnicas integrantes del método.

Por último un *modelo* es una representación abstracta de entes o fenómenos de la realidad en la que se consideran los aspectos relevantes de los mismos y se desechan los menos relevantes, sin que por ello deje de representar significativamente esta realidad. Un modelo constituye por tanto un producto final, y en nuestro caso será generado como resultado de la ejecución de una o varias

actividades. Los modelos además definen una serie de normas en la forma de uso del lenguaje de modelado, y por tanto en la forma de ejecutar la actividad.

Una vez clarificada la terminología, a continuación presentamos cómo se integra cada uno de estos conceptos en el ámbito de la Ingeniería Web.

1.3.2 Ingeniería Web y el proceso de desarrollo del software

El primer paso para sistematizar el desarrollo de aplicaciones web complejas, con una alta necesidad de evolución, extensibilidad y mantenimiento, es definir un proceso de ciclo de vida acorde para ellas. Algunas características deseables de este proceso es que sea altamente iterativo e incremental, con el objetivo de disminuir el tiempo de desarrollo y mejorar la calidad del producto software final. Numerosas aproximaciones abogan en este sentido por un proceso en espiral, propuesto originariamente por (Boehm, 1976). Una posible adaptación al ciclo de vida de las aplicaciones web de este proceso puede ser vista en la Fig. 1.1 (R.S. Pressman, 2000).

En esta figura se observa cómo el proceso de ciclo de vida comienza con una etapa de *Comunicación con el cliente*, que identifica las metas y objetivos de la aplicación web. A continuación, durante la fase de *Planificación*, se estima el coste global del proyecto y se evalúan los riesgos asociados con el esfuerzo de desarrollo. En la fase de *Análisis de requisitos* se establecen los requisitos técnicos para la aplicación web y se identifican los elementos de contenido que se van a incorporar. También se definen los requisitos de diseño gráfico (estética). La actividad de *Ingeniería* incorpora dos flujos de tareas paralelas. Por un lado se efectúa el diseño del contenido (texto, imágenes, vídeos, etc.). Por otro, se debe diseñar la estructura y comportamiento de la aplicación en sí, incluyendo arquitectura, navegación e interfaz entre otras características. La integración de los resultados de ambos flujos conforma el artefacto de salida resultado de esta actividad. Durante la actividad de *Construcción y Adaptación* no sólo se implementan (de ma-

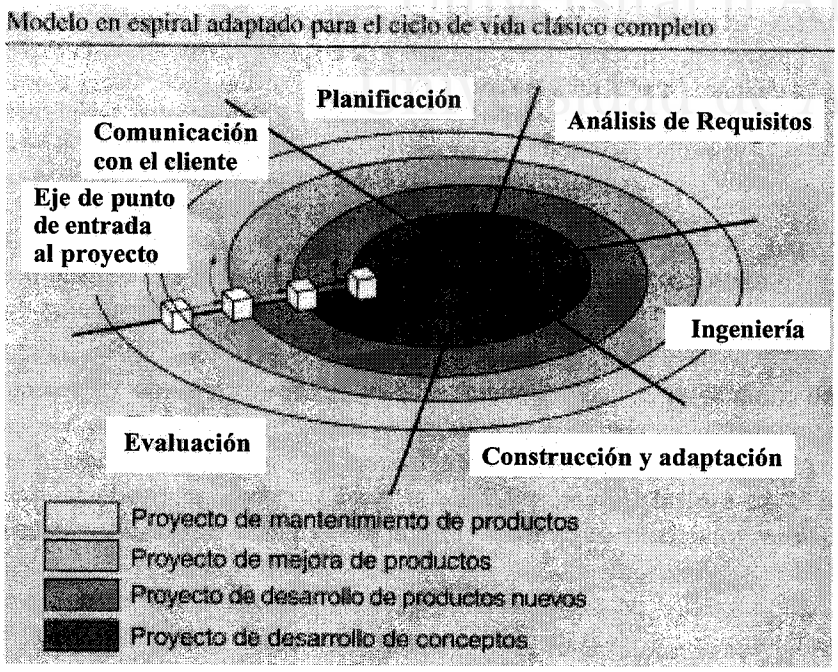

Figura 1.1. Proceso de desarrollo en la Ingeniería Web

nera automatizada o no) los distintos módulos que componen el producto software sino que también se lleva a cabo la integración con software intermedio. Por último, en la fase de *Evaluación* se efectúa la verificación (comprobación de que la aplicación hace lo que el diseñador esperaba que hiciese) y validación (comprobación de que la aplicación hace y tiene la apariencia que el cliente esperaba) de la aplicación. Es en esta fase donde se producen los incrementos de ámbito, que constituyen la entrada para la siguiente iteración, siguiendo un flujo incremental de proceso. No todas las actividades tienen la misma importancia en todas las fases (ciclos de la espiral), y de hecho algunas de ellas ni siquiera tienen por qué estar presentes en algunas de dichas fases.

Es importante observar cómo se puede trazar un paralelismo entre las actividades planteadas en la Fig. 1.1 y las planteadas en otros modelos de proceso bien conocidos, como es el caso del Pro-

ceso Unificado de Rational (RUP). A modo de ejemplo, en este último se plantea una serie de actividades de apoyo (manejo de configuración y cambio, manejo de proyecto y entorno) que en la Fig. 1.1 equivale a las actividades de comunicación con el cliente y planificación. La actividad de análisis de requisitos se mantiene, y a la actividad RUP de análisis y diseño se la llama aquí ingeniería. Por último, la actividad de implementación del RUP equivale a la actividad de construcción y adaptación de la Fig. 1.1. El flujo paralelo de manejo de calidad del RUP se engloba aquí en la actividad de evaluación.

También en las fases del proceso (desarrollo de conceptos, desarrollo de productos nuevos, mejora de productos y mantenimiento, ver Fig. 1.1) se puede trazar un paralelismo con las fases de incepción, elaboración, construcción y transición presentes en el RUP.

1.3.3 El proceso de autoría de las aplicaciones web

Dentro del ciclo de vida de las aplicaciones web, que comienza cuando se concibe la idea de un sistema hipermedial y acaba cuando el producto deja de estar en uso, existe un subciclo denominado *ciclo de autoría*, que abarca desde la decisión de construir el sistema hasta que la aplicación es entregada (Koch *et al.*, 2001).

En general, y como en otros temas relacionados con la Ingeniería Web, la definición de este proceso de autoría no parte de la nada. Existen propuestas de adaptación y extensión de procesos de desarrollo para aplicaciones estándares (como son el Proceso Unificado de Modelado de Rational (RUP, 2001) o ICONIX (Rosenberg & Scott, 1998)) para su aplicación al ámbito hipermedial (Conallen, 1999; Koch *et al.*, 2001). Sin embargo la heterogeneidad de las aplicaciones que encontramos en Internet, y que vimos en la sección 1.2.1, dificulta el consenso en cuanto a cuál es la configuración ideal de proceso asociado a la fase de autoría del producto software.

En efecto, el grado en que los distintos factores (ver sección 1.2.2) afectan a los distintos tipos de aplicaciones web varía ampliamen-

te, y determina el conjunto y el peso de las actividades que deben ser incluidas en dicho proceso de desarrollo. De hecho algunos autores (Ginige, 1998) incorporan como parte del ciclo de autoría de la aplicación una actividad de *modelado de proceso* en la que el objetivo primordial es adaptar el proceso genérico a la idiosincrasia tanto de la aplicación considerada como de la organización en el seno de la cual se desarrolla. En este sentido nos gustaría destacar la propuesta de marco de proceso conocida como Web-OPEN, que supone una adaptación al entorno web de la propuesta metodológica de carácter general OPEN (Lowe & Henderson-Sellers, 2001). Este marco de proceso es un metamodelo a partir del cual se puede instanciar un proceso específico adaptado al tipo de organización y aplicación de que se trate, y cuyo estudio queda fuera del ámbito del presente trabajo.

En relación con el proceso, en este trabajo nos hemos centrado en el estudio de las actividades *de autoría*, que engloban desde el análisis de requisitos hasta la evaluación del cliente (ver Fig. 1.1). El resultado de dicho estudio ha sido un método que, dentro del proceso de desarrollo global, incluye la notación, los modelos y las técnicas necesarias para capturar los aspectos relevantes de la fase de autoría de una aplicación web e integra, siguiendo la estela de diversos referentes en el ámbito hipermedial (Ginige, 1998; Conallen, 1999; Koch *et al.*, 2001; Lowe & Henderson-Sellers, 2001), un subproceso con las mismas características que el proceso del ciclo de vida (iterativo e incremental) para el desarrollo de las distintas tareas que lo conforman, especialmente aquéllas relacionadas con la actividad de ingeniería (análisis y diseño) de la aplicación.

1.4 Uso de técnicas para el desarrollo de aplicaciones web

Además de la definición de un proceso sistemático y repetible, la Ingeniería Web reconoce la necesidad del proporcionar técnicas de modelado que sustenten cada una de sus actividades clave. A pesar de ello, aún no se ha encontrado respuesta a muchas de

las cuestiones referentes al modo de materializarlas, i.e., qué tipos de modelo, técnicas y herramientas se deberían desarrollar para darles soporte; más aún, ni siquiera existe un acuerdo en cuanto a cuáles deberían ser estas actividades clave. Cuestiones como ¿es necesario un modelo de tareas? o ¿cómo se define un modelo de diálogo usuario-sistema, y a qué nivel de detalle debe llegar? ¿Es necesario un modelo de navegación independiente de dispositivo? ¿Qué papel juegan los componentes software, y en qué fase del proceso de autoría deberían ser considerados? siguen causando controversia. Uno de los motivos de esta situación es la existencia de distintas perspectivas a la hora de abordar este proceso de autoría, en función de la formación previa de los analistas/diseñadores.

1.4.1 Perspectivas de modelado de las aplicaciones web

La primera comunidad embarcada en el proceso de plantear un método de modelado para este tipo de aplicaciones fue la comunidad hipermedial. En sus propuestas iniciales se concibió el concepto de aplicación web como un grafo dirigido, donde cada nodo correspondía a una página y el esquema de navegación lo constituía la red de conexiones entre nodos.

Sin embargo esta concepción evolucionó rápidamente gracias a la influencia de grupos provenientes del mundo de las bases de datos, que, ante los requisitos por parte de las empresas de migrar la información contenida en sus repositorios de información a la web, dotaron al concepto de aplicación web de la dinamicidad que requería su nuevo papel. A pesar de esto, las necesidades de funcionalidad consideradas en este tipo de aproximaciones, claramente orientadas a los datos, se seguían limitando a operaciones CRUD.

Con el auge de las tecnologías Orientadas a Objetos y las necesidades causadas por la implantación masiva de las aplicaciones web, nuevos grupos, la mayoría provenientes del campo de la Ingeniería del Software, han ido un paso más allá y han propuesto la generalización del proceso de desarrollo de una aplicación web

y su asimilación al de cualquier otro tipo de aplicación software distribuida con funcionalidad compleja (Manola, 1999). Las soluciones aportadas por este nuevo grupo de aproximaciones suele implicar la extensión de métodos de modelado conceptual tradicionales con nuevos modelos que se ocupen explícitamente de la idiosincrasia de la web.

Por último, los recientes avances en el campo de los componentes distribuidos también han tenido repercusión en el modelado de aplicaciones web, dando lugar a un nuevo enfoque que concibe este entorno de trabajo como una web de negocio dinámica (Gartner Group, 2001), donde la fase de desarrollo implica un proceso de comunicación e integración de servicios diseminados en la red y ofertados via (a menudo) especificaciones de amplia difusión como son UDDI[14], DSML[15], SOAP[16] o WSDL[17]. Estas propuestas dejan por tanto en un segundo plano el tratamiento de la parte informacional de las aplicaciones web, y se centran en otros aspectos, como son los relacionados con la seguridad, la gestión de transacciones, etc.

1.4.2 Necesidad de modelos y técnicas

Sin embargo, sea cual sea el origen de las propuestas, y del mismo modo que ha ocurrido en aproximaciones de desarrollo de software tradicional, todas ellas reconocen la necesidad de desacoplar hasta donde sea posible las distintas perspectivas desde las que se puede considerar una aplicación web, así como de retrasar lo más posible la adopción de decisiones de diseño basadas en consideraciones relativas tanto al entorno de implantación como a arquitecturas, plataformas y tecnologías de implementación.

En este sentido el uso de modelos conceptuales proporciona un mecanismo útil para eliminar ambigüedades y reflexionar sobre la manera más apropiada de abordar un determinado problema.

[14] Universal Description, Discovery and Integration (UDDI, 2001)
[15] Directory Services Markup Language (DSML, 2000)
[16] Simple Object Access Protocol (SOAP, 2001)
[17] Web Services Description Language (WSDL (WSDL, 2001))

Además el uso de modelos ayuda en las tareas de comunicación y documentación del sistema, y favorece tanto la división del problema como la repartición de tareas. Otra ventaja importante es que el nivel de abstracción de estos modelos facilita la reutilización de los mismos ante cambios en el entorno.

Dimensiones de modelado de las aplicaciones web. En el ámbito de las aplicaciones web, la propuesta de modelos conceptuales se ha vertebrado en torno a tres conceptos principales (Retschitzegger & Schwinger, 2000) (ver Fig. 1.2):

- **fases**: niveles de abstracción que deberían tener soporte en cualquier propuesta de modelado para favorecer tanto la reflexión como el reuso: (1) un análisis o modelado conceptual (representación abstracta del dominio), (2) un diseño o modelado lógico (diseño independiente de tecnología), (3) una fase de modelado físico (diseño dependiente de tecnología) y (4) una fase de implementación.
- **niveles**: representan los distintos puntos de vista desde los cuales se puede estudiar el sistema: contenido, navegación (hipertexto) y presentación.
- **aspectos**: añade la perspectiva acerca del carácter estático (estructural) /dinámico (comportamiento) de cada uno de los niveles. A nivel de contenido, la parte estática se obtiene a partir de un proceso de *clasificación* de los conceptos de dominio, mientras que la parte dinámica la constituyen las operaciones asignadas a cada uno de esos elementos de dominio. A nivel de navegación, la definición de caminos entre objetos de dominio refleja su parte estática, mientras que el modelado de las condiciones que posibilitan la generación/ocultación dinámica de enlaces y/o el cambio en los resultados de navegar a través de los mismos en función de parámetros de ejecución constituye su aspecto dinámico. Por último, a nivel de presentación, la visualización de las páginas en términos de estilos, organización visual etc. constituye su parte estática, y las reacciones ante eventos de usuario, la interacción y sincronización entre elementos de interfaz, etc. constituyen su faceta dinámica.

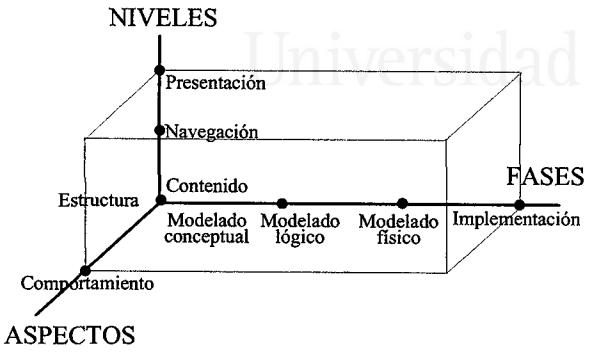

Figura 1.2. Dimensiones de Modelado de las aplicaciones web

Fases, niveles y aspectos son ortogonales entre sí, y su combinación (dejando de lado la fase de implementación) da lugar a 18 posibles perspectivas de la aplicación: desde una perspectiva conceptual de la estructura de dominio (información estática de dominio) hasta una perspectiva a nivel físico del comportamiento de la presentación (e.g. respuesta, dependiente de plataforma, de los elementos físicos de la interfaz ante eventos). Sin embargo, la complejidad de uso de un método exhaustivo que incluya tal cantidad de perspectivas lo hace inviable a nivel práctico. Así las aproximaciones al modelado hipermedial existentes en la actualidad se centran en la especificación, mediante un pequeño número de modelos, de un subconjunto de estas perspectivas, que experiencias empíricas han demostrado suficientes para la captura de los aspectos más relevantes de la mayor parte de las aplicaciones web actuales. Dentro de este subconjunto, (1) el modelado conceptual de la estructura y comportamiento de dominio, (2) el modelado lógico de estructura y comportamiento de navegación y (3) el modelado lógico de la estructura de la presentación son elementos comunes en las propuestas hipermediales que han ad-

quirido una mayor difusión, tal y como veremos en el capítulo 2.

1.4.3 Influencia de las técnicas de calidad en las aplicaciones resultantes

Por último, no queremos dar por concluida la sección sin comentar brevemente los mecanismos de calidad que ofrecen las propuestas de modelado que abordan el desarrollo de aplicaciones web desde un punto de vista ingenieril. De hecho, si bien es cierto que las actividades orientadas a garantizar la Calidad del Software[18] deben estar presentes en todo el ciclo de desarrollo del software, esta afirmación es especialmente relevante en el caso de aplicaciones web, debido a la baja tolerancia (comentada en la sección 1.2.2) de los usuarios finales a errores o interfaces difíciles de usar.

El primer mecanismo lo constituye la propia existencia de un proceso de desarrollo de software visible, repetible y medible (como los presentados en la sección 1.3.1.), que se consolida así como el primer paso hacia un Manejo Total de la Calidad[19].

El segundo paso es la definición de actividades, técnicas y herramientas que, integradas dentro de este proceso de desarrollo, contribuyen al incremento de calidad global de aplicaciones web. Para ello es necesario previamente definir qué es 'calidad de software' a distintos niveles de abstracción. En el contexto de este trabajo distinguimos dos tipos de calidad (R.S. Pressman, 2000), cada uno de los cuales se ve afectado por el uso de distintos mecanismos:

- Calidad de Diseño. Con este concepto englobamos calidad en recogida de requisitos, especificación y diseño de la aplicación. En este punto juegan un papel importante los **patrones** definidos a distintos niveles de abstracción (e.g. patrones de diseño de Gamma *et al.* (Gamma *et al.*, 1995)): mecanismos de abstracción que permiten el reuso de experiencias de probada eficacia.

[18] SQA: *Software Quality Assurance*
[19] TQM: *Total Quality Management*

El uso de patrones disminuye la probabilidad de error, acorta el tiempo de desarrollo y garantiza el reuso de soluciones efectivas.

- Calidad de Ajuste. Con este concepto nos referimos al grado en el cual el producto final se ajusta a las especificaciones. En este punto los **compiladores de modelos** (Bell, 1998) juegan un papel importante: automatizando las tareas de producción de software a partir de modelos de diseño aseguramos que la aplicación final tiene el máximo grado de ajuste con respecto a la especificación, al evitar la toma de decisiones arbitrarias, no documentadas, por parte de los programadores.

Aunque la disciplina de evaluación de software tiene más de tres décadas, la evaluación de calidad sistemática y cuantitativa de aplicaciones hipermediales es muy reciente. Dentro de las propuestas existentes, podemos destacar el árbol de requierimientos presentado por Olsina en (Olsina *et al.*, 2001), que personaliza las métricas de calidad en función del tipo de audiencia. Aunque las métricas de calidad de aplicaciones hipermediales quedan fuera del ámbito de este trabajo, nos gustaría destacar cómo la adopción de **estándares** para el diseño de los distintos modelos del proceso facilita la evaluación de los mismos al simplificar la adopción de métricas existentes, ampliamente validadas.

Tal y como veremos en el capítulo 3, patrones, compiladores de modelos y uso de estándares son precisamente tres de los rasgos que caracterizan la aproximación presentada en el presente trabajo.

1.5 Objetivos de la presente tesis

En base a lo comentado a lo largo del capítulo, podemos concluir que el objetivo general de este trabajo no ha sido definir un método más de modelado de interfaces hipermediales, sino integrar los conceptos que consideramos más significativos de otras propuestas existentes en un marco que, además, incluye nuevas ideas orientadas a la resolución de los problemas y retos actuales

en el campo de desarrollo de las aplicaciones web, y que han sido detectados como resultado de un estudio exhaustivo que presentaremos en el capítulo 2. Estas aportaciones se pueden sintetizar en:

- Un proceso sistemático de diseño que guía el desarrollo de interfaces hipermediales. Este proceso se verá en el capítulo 3.
- Un marco conceptual para la representación del espacio de navegación de cada tipo de usuario (vistas de información, caminos de navegación entre vistas y restricciones, estáticas o dinámicas, que le son aplicables), donde destaca el uso de un lenguaje estandarizado para especificar las restricciones de navegación. Este marco será abordado en los capítulos 4 y 5.
- Una actividad explícita de modelado de interfaces de servicio con capacidad de interconexión con módulos de funcionalidad complejos, no limitados a servicios de altas, bajas y modificaciones. Esta propuesta, basada en una taxonomía de tipos de parámetro que definen el modo en que el usuario puede introducir los valores con los que se invocará al método, será ilustrada en el capítulo 6.
- Un conjunto de mecanismos que, integrados en el resto de modelos, facilitan la personalización de aplicaciones. La personalización será tratada en el capítulo 7.
- Un modelo lógico de presentación en sus dos vertientes: estructural (modelo lógico del sistema) y de diseño visual (aspecto de páginas independientes de dispositivo y/o plataforma objetivo), que puede ser refinado mediante la aplicación de un lenguaje de patrones, promoviendo de este modo el reuso de experiencias de diseño. Este aspecto será abordado en el capítulo 8.
- Una herramienta CAWE[20] que da soporte a la propuesta e incluye un potente entorno de prototipado a partir de los modelos OO-H. Este entorno viene acompañado de un compilador de modelos que, a partir de una especificación independiente de plataforma y de dispositivo, genera de manera automática la

[20] *Computer Aided Web Engineering*

aplicación tomando en consideración parámetros referentes al entorno de implementación. Este aspecto será abordado en el capítulo 9.

Además, a la hora de abordar estos objetivos hemos impuesto las siguientes restricciones:

- Uso de estándares que faciliten la integración de/con otras técnicas, herramientas, metodologías y/o métodos.
- Definición de procesos de transformación entre modelos que disminuyan su grado de acoplamiento y faciliten la trazabilidad de los distintos conceptos que intervienen en el diseño de una aplicación web.
- Separación, a la hora de obtener la especificación lógica de interfaz que se deriva de los modelos propuestos, de los distintos aspectos que intervienen en su definición (mediante el uso de plantillas de distintos tipos), para facilitar su mantenimiento.
- Almacenamiento y automatización de los mecanismos de reuso como parte del método.

Para la materialización de estas aportaciones, se ha seguido un método de investigación cualitativo, denominado *Investigación en Acción*[21], que presentamos a continuación.

1.6 Método de investigación

La investigación en acción puede ser definida como *el proceso de recopilar de forma sistemática datos de la investigación acerca de un sistema actual en relación con algún objetivo, meta o necesidad de ese sistema; de alimentar de nuevo con esos datos al sistema; de emprender acciones por medio de variables alternativas seleccionadas dentro del sistema, basándose tanto en los datos como*

[21] *Action Research*

en las hipótesis, y de evaluar los resultados de las acciones, recopilando datos adicionales (French & Bell, 1996). En general, los métodos de investigación cualitativos se han demostrado apropiados para el campo de los sistemas de información (Avison *et al.*, 1999).

1.6.1 Participantes

En este tipo de método de investigación participan *todas las partes involucradas en la investigación, examinando la situación existente (que sienten como problemática), con los objetivos de cambiarla y mejorarla* (Y. Wadsworth, 2001). Existen cuatro tipos de actores: el investigador, el objeto investigado, el grupo crítico de referencia (personas para las que se investiga y que participan, sabiéndolo o no, en el proceso de investigación) y entes externos a la investigación pero que se benefician de sus resultados.

En nuestro caso estos actores han sido los siguientes:

- Grupo investigador: grupo de Ingeniería Web de la Universidad de Alicante (a partir de ahora WebE-I).
- Objeto investigado: método de desarrollo de aplicaciones hipermediales.
- Grupo crítico de referencia: (1) parte del propio grupo de Ingeniería Web de la Universidad de Alicante (a partir de ahora WebE-GC), que ha planteado aplicaciones ficticias sobre las que ha realizado una evaluación preliminar de la capacidad expresiva del método, y (2) la empresa SUMA Gestión Tributaria (a partir de ahora SUMA), que ha proporcionado especificaciones reales sobre las que se ha podido constatar la validez del método propuesto en el presente trabajo.
- Beneficiario de la investigación: cualquier organización que sea cliente o proveedor de aplicaciones centradas en el manejo de datos y que estén basadas en web.

1.6.2 Tipo de Investigación en Acción

Por otro lado, se pueden distinguir varios tipos de Investigación en Acción (French & Bell, 1996):

- De diagnóstico: se realizan recomendaciones para la resolución de un problema, pero sin una evaluacón previa ni un control posterior.
- Participante: el grupo crítico de referencia trabaja con el investigador, poniendo en práctica sus recomendaciones y compartiendo sus resultados.
- Empírica: el grupo crítico de referencia registra de manera exhaustiva sus acciones y sus efectos.
- Experimental: el investigador evalúa distintas maneras de conseguir un objetivo.

El tipo de investigación en acción utilizada en la elaboración del presente trabajo ha sido la *Investigación en Acción participante*, en la que, ante el problema de cómo modelar de forma efectiva los rasgos más relevantes de las aplicaciones web encontradas en Internet, hemos diagnosticado y realizado recomendaciones en forma de un método de modelado hipermedial conocido como OO-H. Este método ha sido aplicado para la resolución de problemas reales por parte de diversos grupos críticos de referencia (componentes del Grupo de Ingeniería Web de la UA, SUMA), y han producido una retroalimentación hacia el investigador, que ha servido para refinar el método en iteraciones sucesivas.

La implementación del método de trabajo ha seguido en lo posible los pasos recomendados por (Padak & Padak, 1994), y que son:

- Identificar las cuestiones que deben guiar la investigación, que deben ser relevantes, estar directamente relacionadas con el objeto que se está investigando y ser susceptibles de encontrarles respuesta.
- Recoger información relacionada con esas cuestiones (bibliografía, entrevistas, observaciones, etc.)

- Analizar la información recogida.
- Compartir los resultados con el resto de los interesados, con el fin de plantear nuevas cuestiones relevantes y profundizar en el conocimiento adquirido.

A continuación veremos los principales hitos cubiertos.

1.6.3 Hitos de la investigación

La investigación que ha desembocado en el presente trabajo ha sido realizada en su mayor parte con la ayuda de una beca de Formación del Personal Investigador (FPI) concedida por la Conselleria de Cultura, Educació y Ciència a la autora del mismo. Es por ello que la identificación de cuestiones que debían ser resueltas en cada etapa de la investigación ha sido cuidadosamente planificada y recogida en tres informes correspondientes a los años 1999 (Cachero, 1999), 2000 (Cachero, 2000)y 2001 (Cachero, 2001). Esta planificación ha servido de guía para la recogida de información relevante, el análisis de dicha información y la compartición de resultados (1) con el resto del equipo del Grupo de Ingeniería Web de la Universidad de Alicante (mediante reuniones e informes internos que han contribuido al refinamiento de OO-H y a la implementación de la herramienta que da soporte al método) y (2) con el resto de la comunidad científica (mediante la publicación y presentación de artículos en congresos y revistas nacionales e internacionales, tal y como detallaremos en el capítulo 10). De entre estas actividades de difusión, destaca la participación en el taller IWWOST[22] que, tras dos ediciones, se ha erigido en foro de discusión donde representaciones de las propuestas hipermediales más relevantes lanzan nuevos retos e ideas para seguir avanzando en el ámbito del modelado hipermedial.

A continuación desglosamos la actividad efectuada durante el período de investigación que ha dado lugar al presente trabajo.

[22] *International Workshop on Web Oriented Software Technology*

Módulo	Tarea	Hito	Participantes		
			WebE-I	WebE-GC	SUMA
M0: Planificación tareas	T1	Planteamiento inicial del problema	☑	☐	☐
	T2	Planificación de la estrategia de trabajo para abordarlo	☑	☐	☐
M1: Análisis del estado del arte en métodos hipermediales	T1	Recopilación de las características que han de ser tenidas en cuenta para el diseño de un método de desarrollo de aplicaciones hipermediales mediante el análisis de aplicaciones reales	☑	☐	☐
	T2	Estudio de las aproximaciones hipermediales propuestas en el ámbito de la comunidad científica	☑	☐	☐
M2: Formación avanzada en metodologías de análisis y diseño	T1	Estudio de las características principales de distintas propuestas notacionales formales y semiformales para el desarrollo de aplicaciones de carácter general (UML, OO-Method).	☑	☐	☐
M3: Elaboración de una propuesta: el método OO-H	T1	Definición de semántica del hipertexto: identificación de correspondencias entre expresividad en el espacio del problema y expresividad en el espacio de la solución.	☑	☐	☐

Figura 1.3. Principales Hitos de la Investigación. Año 1999.

Año 1999. Tal y como podemos observar en la Fig. 1.3, durante el primer año (1999) la investigación se centró, una vez definido el problema[23] en la documentación del equipo investigador mediante el análisis de aplicaciones reales y el estudio de las propuestas que desde la comunidad científica se habían realizado para darles solución.

La heterogeneidad de las propuestas y notaciones estudiadas nos impulsó a buscar consenso por medio del uso de notaciones estándares, para lo cual abordamos el estudio de propuestas como UML, al mismo tiempo que intentábamos identificar aquellos puntos del proceso en que su inclusión resolvía de forma natural los distintos subproblemas planteados. Paralelamente, se trabajó en la abstracción de un conjunto limitado de conceptos semánticos particulares al ámbito hipermedial. Estos conceptos debían proporcionar una respuesta genérica y flexible a los distintos problemas de modelado de interfaces hipermediales, y no circunscribirse a una situación de modelado concreta en una aplicación particular.

El resultado de este trabajo, en su mayoría de documentación, consistió en un conjunto de informes internos (uno sobre el esta-

[23] La definición preliminar de dicho problema se basó en la experiencia previa de la autora como programadora de aplicaciones web, aspecto que contribuyó a la inclusión de objetivos, poco comunes en las propuestas existentes en aquel momento, como la migración de bases de datos preexistentes, la conexión con módulos de lógica o el diseño de un lenguaje de patrones que permitiese acelerar el proceso de desarrollo y asegurar la homogeneidad en el tratamiento de las distintas condiciones de interacción usuario-sistema.

do de la investigación de aplicaciones hipermediales en el ámbito científico, otro sobre la propuesta notacional y semántica de OO-H y un tercero sobre los patrones de interfaz que iban siendo detectados) que sirvieron de base para publicaciones subsiguientes.

Módulo	Tarea	Hito	Participantes		
			WebE-I	WebE-GC	SUMA
M2: Formación avanzada en metodologías de análisis y diseño	T2	Estudio de los posibles modos de acoplamiento entre módulos de lógica y módulos de interfaz en un entorno de generación automática de software	☑	☐	☐
M3: Elaboración de una propuesta: el método OO-H	T2	Elaboración de una propuesta de análisis y diseño de navegación que incluye un conjunto limitado de constructores: el diagrama de acceso navegacional (DAN).	☑	☑	☐
	T3	Refinamiento de la semántica del diagrama de navegación y diseño de presentación: el diagrama de presentación abstracta (DPA) y el Diagrama de Diseño Visual (DDV)	☑	☑	☐
	T4	*Descripción estandarizada de las aplicaciones OO-H: uso de* plantillas XML.	☑	☐	☐
	T5	Identificación de patrones de interfaz que documenten y aceleren el proceso de desarrollo en OO-H	☑	☑	☐
M4: Compilador de modelos y generación automática de código en OO-H	T1	Estudio sobre la viabilidad de distintas técnicas de integración de los componentes del modelo objetual para la generación de código.	☑	☐	☐
	T2	Prototipado del algoritmo de generación automática de código en OO-H	☑	☐	☐
M5: Generalización del compilador de modelos, integración de patrones y evaluación de la calidad de la interfaz generada	T1	Comparativa con otros métodos y entornos de desarrollo hipermedial: expresividad semántica y apariencia final de interfaz.	☑	☑	☐

Figura 1.4. Principales Hitos de la Investigación. Año 2000.

Año 2000. Una vez concluido el período de documentación exhaustiva, durante el año 2000 (ver Fig. 1.4) el conocimiento adquirido se plasmó en una propuesta de diagramas de navegación y presentación. Los conceptos englobados en esta propuesta siguen hoy en día vigentes y conforman en núcleo de OO-H. Además, se planteó una primera taxonomía de plantillas para describir de manera textual la interfaz modelada, y se propuso la inclusión de un lenguaje de patrones para agilizar y documentar las decisiones de diseño adoptadas. Paralelamente, se abordó el estudio de posibles modos de interconexión automática entre la interfaz y módulos de lógica subyacente, con el fin de testar la viabilidad de nuestro objetivo.

Por otro lado, la necesidad de generación de código a partir de los modelos OO-H nos hizo plantearnos un estudio sobre distintas técnicas de implementación y arquitectura tanto del entorno de desarrollo como de los compiladores de modelos, con el fin de garantizar su flexibilidad ante condiciones cambiantes del método y/o de los requisitos de las aplicaciones modeladas.

Por último, la autora del presente trabajo se desplazó durante tres meses al Politécnico de Milán, donde, bajo la tutela del Dr. Stefano Ceri y del Dr. Piero Fraternali profundizó en el conocimiento de WebML (desde nuestro punto de vista, una de las propuestas más maduras y viables del panorama de modelado hipermedial actual) y trabajó en una comparativa sobre la expresividad semántica de ambas aproximaciones.

El resultado de este trabajo se vio plasmado en cuatro artículos aceptados en congresos internacionales, un artículo aceptado en un congreso nacional y varios informes internos, entre los que destacamos una ampliación y sistematización del Catálogo de Patrones de OO-H.

Módulo	Tarea	Hito	Participantes		
			WebE-I	WebE-GC	SUMA
M2: Formación avanzada en metodologías de análisis y diseño	T2	Estudio de los posibles modos de acoplamiento entre módulos de lógica y módulos de interfaz en un entorno de generación automática de software	☑	☐	☐
M3: Elaboración de una propuesta: el método OO-H	T6	Definición de la evolución de los distintos diagramas ante la aplicación de cada uno de los patrones recogidos en el Catálogo de Patrones de Interfaz	☑	☐	☐
M4: Compilador de modelos y generación automática de código en OO-H	T3	Diseño e implementación del entorno de desarrollo y generación de interfaces	☑	☑	☐
M5: Generalización del compilador de modelos, integración de patrones y evaluación de la calidad de la interfaz generada	T2	Implementación de los algoritmos de generación de componentes de interfaz para entornos industriales de desarrollo a partir de la	☑	☑	☐
	T3	Aplicación del entorno de programación automática ampliado a la resolución de sistemas de información complejos reales	☑	☑	☑
	T4	Estudio de la bondad de las interfaces generadas siguiendo nuestro método de modelado	☑	☑	☑

Figura 1.5. Principales Hitos de la Investigación. Año 2001.

Año 2001. Durante el 2001 se siguió trabajando en el modo de acoplar las interfaces generadas con bases de datos y módulos

de lógica subyacentes, y se estudiaron estándares como son los relacionados con los Servicios Web. Además, y paralelamente al enriquecimiento y maduración de los conceptos planteados en años anteriores, se incluyó una nueva sección dentro de cada uno de los patrones propuestos para el Catálogo de Patrones de OO-H, que permite, mediante una notación propietaria, la aplicación automática de dichos patrones a los distintos modelos, acelerando de este modo el proceso de desarrollo.

Es también en este año cuando se presentó la primera versión estable de la herramienta CAWE que da soporte al método, conocida como Visual WADE[24]. Esta primera versión fue un entorno principalmente diagramático, aunque a lo largo del año se fueron incluyendo algoritmos de generación restringidos a un subconjunto de los rasgos de interfaz capturados en los modelos. Además, muchas características adicionales del entorno actual ya estaban presentes en esa primera versión: es el caso del entorno de prototipado o la estrategia de manipulación de la descripción XML subyacente.

Con esta primera herramienta se procedió al desarrollo parcial de interfaces correspondientes a sistemas de información reales, como puede ser un Sistema de Gestión de Hoteles o un Sistema de Gestión de Conferencias. También durante este año se firmaron los primeros convenios entre la universidad y empresas, entre las que destacamos la experiencia con SUMA, que nos ha proporcionado no sólo valiosos casos de estudio sino que han actuado como Grupo Crítico de Referencia, enriqueciendo con sus continuos comentarios tanto la expresividad de los constructores del método como las capacidades de generación del entorno. Además, SUMA nos proporcionó, ya desde las primeras interacciones, una evaluación informal de la calidad de las interfaces generadas[25].

El resultado de este trabajo se vio plasmado en un artículo en la revista IEEEMultimedia, un artículo en un congreso internacio-

[24] emphVisual Web Application Development Environment (Entorno Visual de Desarrollo de Aplicaciones Web)

[25] Un estudio formal exhaustivo de los parámetros de calidad de las interfaces generadas mediante OO-H queda propuesto como trabajo futuro.

nal, dos artículos en congresos nacionales y varios informes internos, entre los que destacan los relativos a la especificación XML de las interfaces modeladas mediante OO-H.

Año 2002. Durante este último año se ha continuado buscando la colaboración con empresas, que han dado lugar a numerosas experiencias de modelado de aplicaciones reales, entre las que podemos citar:

- Una web educativa (E-Aula)
- Una gestión de inventario para SUMA
- Una simulación de la aplicación CAM directo
- Un sistema de gestión de conferencias revisado

A nivel de entorno de desarrollo, el trabajo se ha centrado en la depuración de errores y el incremento de la facilidad de uso, así como en la construcción y depuración de los compiladores de código que lo acompañan. Además, se ha desarrollado un entorno de compilado externo ubicado en la URL *http://www.visualwade.ua.es* que permite enviar por internet proyectos elaborados en el seno de Visual WADE, compilarlos en remoto y poder navegar por la aplicación generada como resultado de dicha generación, evitando de este modo que los diseñadores tengan que instalar un entorno de ejecución en sus puestos de trabajo locales para poder validar los modelos. El lector interesado puede conectarse a dicha dirección para visualizar algunos ejemplos de modelos y las aplicaciones generadas a partir de ellos.

1.7 Resultados conseguidos

El presente trabajo ha cubierto todos los objetivos planteados en la sección 1.5 y proporciona, como contribución básica, un conjunto de constructores y modelos para capturar los aspectos relevantes involucrados en el modelado e implementación de interfaces para aplicaciones web. De entre estos constructores destaca

la potencia semántica del *Enlace Navegacional*, que permite el modelado de situaciones comunes en aplicaciones reales y no consideradas en otras propuestas de modelado, como puede ser la selección de varios objetos sobre los que se desea aplicar una misma acción de navegación. Dentro de este concepto, el *Enlace de Servicio* permite el modelado de procesos de recogida de parámetros y visualización de parámetros de retorno de métodos subyacentes, independientemente de que estén o no incluidos como parte de una base de datos. Este aspecto también ha sido obviado tradicionalmente en la mayoría de las aproximaciones hipermediales existentes.

Los mecanismos de especificación de restricciones a través del espacio de información proporcionados por OO-H (en base a un lenguaje de especificación de restricciones estándar), unido a su modelo de referencia, permiten no sólo el modelado de caminos navegacionales complejos sino también la implantación de políticas de personalización no triviales. Además, este trabajo presenta un conjunto de patrones de interfaz, y propone como novedad principal la inclusión como parte de su especificación de una *regla de transformación* que permite su aplicación directa sobre los modelos, acelerando de este modo el proceso de desarrollo a la vez que garantiza un mayor nivel de homogeneidad y calidad de las decisiones de diseño adoptadas.

Por último, pensamos que el entorno de desarrollo y el conjunto de compiladores que dan soporte a la propuesta de modelado presentada en este trabajo avalan su potencial. Este entorno de desarrollo proporciona además numerosos mecanismos que se han demostrado vitales para la implantación de un método como OO-H, entre los que destacamos su potente entorno de prototipado o la facilidad de extensión que proporciona su arquitectura.

El resultado es un proceso de construcción ágil que produce una especificación de interfaz independiente del dispositivo. A partir de ella una interfaz web, integrable con módulos de lógica preexistentes, puede ser generada de manera automática.

1.8 Estructura del trabajo de investigación

Siguiendo el orden de los objetivos planteados en la sección 1.5, el trabajo está organizado como sigue:

Capítulo 2: Estado de la Investigación en Modelado Web

Este capítulo presenta las distintas tendencias y aproximaciones en el ámbito del modelado hipermedial. Comienza con una breve introducción que nos acerca a sus orígenes, para a continuación revisar las aproximaciones más relevantes que a lo largo de estos últimos años han sido propuestas en la comunidad científica. Posteriormente se analizan su grado de cobertura ante un conjunto de aspectos no triviales, como es el grado de soporte a la personalización, la invocación de servicios o el soporte CAWE. Finalmente se analiza el uso de las aproximaciones conceptuales en los entornos de trabajo actuales.

Capítulo 3: Introducción a OO-H: características generales

Este capítulo comienza introduciendo los objetivos de OO-H. A continuación presenta las principales características de OO-H, y cómo integra diversas perspectivas que contribuyen a producir una especificación más completa de la interfaz objeto de modelo. Una descripción de su proceso de autoría, unida a la presentación del conjunto de modelos y diagramas propuestos y el modo en que se les da soporte mediante un entorno CAWE y un compilador de modelos completan esta visión general de la propuesta.

Capítulo 4: Recogida de requisitos y modelado de dominio en OO-H

Este capítulo comienza presentando un caso de estudio que servirá para ilustrar los principales conceptos de OO-H en capítulos subsiguientes. Una vez introducido el ejemplo, el capítulo desarrolla las tres actividades, comunes a la mayoría de métodos hipermediales, con las que da comienzo el proceso de autoría de OO-H: (1) el análisis de requisitos funcionales, que produce como resultado un Diagrama de Casos de Uso que recoge los requisitos funcionales del usuario, y el (2) análisis y (3) diseño de dominio, mediante

las cuales se crea y refina un Diagrama de Clases, que proporciona una vista estática sobre el dominio de la aplicación. Para cada una de estas actividades, el capítulo presenta la semántica de sus constructores, el proceso de elaboración de los diagramas generados como resultado de cada actividad y un ejemplo de aplicación al caso de estudio.

Capítulo 5: El modelo de navegación de OO-H

Una vez completada la recogida de requisitos y el modelado de dominio, este capítulo presenta el concepto de navegación en OO-H, y cómo éste distingue entre navegación semántica y navegación estructural. La navegación semántica es la base sobre la que OO-H define una actividad de análisis de navegación, reflejada en un Diagrama de Navegación Semántica. Esta actividad se ve complementada con una actividad de diseño de navegación en la que un Diagrama de Acceso Navegacional define los caminos navegacionales que puede seguir cada usuario para dar respuesta a sus requisitos funcionales. El capítulo se completa con una discusión en profundidad acerca de los conceptos y modo de llevar a cabo esta actividad de diseño de navegación, que se encuentra en el corazón del método y constituye una de las principales aportaciones del presente trabajo.

Capítulo 6: Modelado de servicios en OO-H

Este capítulo completa la actividad de diseño de navegación introducida en el capítulo 5 con una explicación exhaustiva del modo en que OO-H modela las interfaces de invocación de servicio. Para ello, el capítulo comienza definiendo el concepto de servicio y una posible categorización en base a distintos criterios. A continuación se presenta la semántica asociada al enlace de servicio, principal constructor involucrado en el modelado de estas interfaces. Finalmente, el uso de este enlace se ejemplifica mediante su aplicación al caso de estudio.

Capítulo 7: Modelado de estrategias de personalización en OO-H

Una vez completado el diseño de navegación, este capítulo muestra cómo los distintos modelos pueden ser enriquecidos para capturar

información relativa a las necesidades de personalización de las interfaces hipermediales. Para ello se comienza dando una breve visión del contexto de la personalización: sus técnicas, sus riesgos y una posible categorización de tipos de personalización existentes. A continuación se comenta el papel que actualmente juega la personalización en las aproximaciones de modelado hipermedial en general y en OO-H en particular. El capítulo se completa presentando las modificaciones que introducen las consideraciones de personalización en el método, y que incluyen (1) modificación del diseño del modelo de referencia, (2) introducción de un *modelo de usuario* y (3) revisión de todas las actividades de OO-H para incluir tareas que tengan en cuenta este nuevo tipo de requisitos.

Capítulo 8: El modelo lógico de presentación en OO-H

Este capítulo presenta la especificación textual que subyace en OO-H, y que actúa de puente entre el espacio del problema y el espacio de la solución, al acercar la especificación a un modelo de implementación web basado en recursos (ficheros). Estos recursos son en realidad instancias de una taxonomía de plantillas, que separan explícitamente contenido, navegación y distintos aspectos de presentación. De entre ellos, los documentos de datos (denominados en OO-H *páginas abstractas*) y el documento de enlazado dan lugar a un Diagrama de Presentación Abstracta, que se encuentra a caballo entre la actividad de diseño de navegación y la actividad de diseño de presentación. Este diagrama, cuyo esqueleto puede ser generado a partir de la información contenida en el DAN, se refina mediante un lenguaje de patrones contenido en un Catálogo que también presentamos en este capítulo. Por último, el capítulo presenta cómo la presentación de cada página abstracta puede ser refinada mediante un Diagrama de Diseño Visual (DDV), hasta conseguir la apariencia deseada para la aplicación final.

Capítulo 9: El soporte CAWE de OO-H

Este capítulo completa nuestra aproximación, y defiende la idea de que, con el fin de facilitar la implantación de métodos y metodologías de modelado hipermedial, es necesario disponer de entor-

nos de desarrollo integrados que les den soporte, y a ser posible integren capacidades de generación de código a partir de modelos. A continuación se presenta Visual WADE como entorno CAWE que da soporte a todas las actividades del proceso de autoría de OO-H, y permite la generación de aplicaciones web operativas. Esta generación se realiza en cuatro fases, y cubre desde la generación de la base de datos a la generación de la documentación. El capítulo finaliza con una breve descripción de una experiencia empírica de implantación de Visual WADE.

Capítulo 10: Conclusiones Finales

En este capítulo se presentan las principales aportaciones y conclusiones derivadas de este trabajo, los trabajos en progreso y líneas futuras de investigación, así como la producción científica que ha generado la investigación.

APÉNDICE A: DTD's correspondientes a la taxonomía de plantillas de OO-H

Este apéndice complementa el capítulo 8 y presenta los DTD's que determinan la sintaxis de los documentos que conforman la especificación textual de las interfaces hipermediales en OO-H.

APÉNDICE B: Un ejemplo de aplicación final generada: Aula Virtual

Este apéndice ilustra la apariencia final de las aplicaciones que pueden ser generadas a partir de los modelos OO-H.

APÉNDICE C: La Autora

Este apéndice resume brevemente la experiencia profesional de la autora del presente trabajo.

2. Estado de la investigación en modelado de aplicaciones web

El poder habita cerca de la necesidad.

PITÁGORAS

Este capítulo presenta las principales tendencias y aproximaciones conceptuales que han sido concebidas para abordar la problemática de modelado de las aplicaciones hipermediales. Para ello, en la sección 2.1 una breve introducción nos acerca a las dificultades encontradas en las aplicaciones hipermediales tradicionales, desarrolladas mediante aproximaciones de manejo de contenido. A continuación, en la sección 2.2 esbozaremos brevemente los distintos frentes desde los que se ha intentado dar respuesta a estos problemas mediante la sistematización de procesos, marcos de referencia, notaciones y modelos. Dentro de estos frentes, en la sección 2.3 destacamos el conjunto de propuestas que, centradas en la definición de actividades y modelos necesarios para reflejar la idiosincrasia de las aplicaciones web, se aproximan más al enfoque adoptado por OO-H. En relación con estas propuestas, un estudio comparativo entre ellas y OO-H, útil para la contextualización de las aportaciones de este último, es presentado en la sección 2.4. Una visión global del estado de implantación de los métodos y metodologías hipermediales (ver sección 2.5) complementa esta comparativa, y cierra el capítulo la sección 2.6, donde se presentan las conclusiones del capítulo.

2.1 Orígenes del desarrollo de aplicaciones web

El desarrollo de aplicaciones web, tal y como comentábamos en el capítulo 1, ha sido considerado durante mucho tiempo como un

problema de autoría de contenidos más que como un problema de ingeniería. Esta concepción ha propiciado el uso extensivo de aproximaciones *ad-hoc* para el desarrollo de aplicaciones web que, ante el aumento de complejidad de las aplicaciones y el conjunto de nuevas responsabilidades que han inducido los nuevos entornos de trabajo, están causando una serie de problemas tanto para el diseñador como para el usuario final. Entre los primeros, relevantes desde el punto de vista del *diseñador*, destacan (Retschitzegger & Schwinger, 2000; Fraternali, 1999):

- *Interdependencia de la lógica de negocio, navegación y presentación* en las aplicaciones. La implementación artesanal favorece las interdependencias entre estructura de datos, navegación y presentación, lo cual conlleva aplicaciones poco flexibles, con procesos de adaptación costosos y complejos (dificultad de personalización, obsolescencia de datos, propagación de errores, imposibilidad de predecir los efectos de los cambios, etc.). En un entorno de rápida evolución tecnológica como es la web, estas limitaciones son aún más graves que en aplicaciones tradicionales, comprometiendo incluso la propia implantación de la aplicación (Fayad & Schmidt, 1997).
- *Falta de mecanismos de comunicación entre los miembros del equipo desarrollador*: la inexistencia de una notación y un vocabulario común dificulta el entendimiento en equipos interdisciplinares, típicos en procesos de desarrollo de aplicaciones web. Este problema se magnifica si tenemos en cuenta que estos equipos se enfrentan a requisitos más inciertos que en el caso de la ingeniería del software tradicional (Atzeni *et al.*, 1998), y causa que el desarrollo de la aplicación se convierta en dependiente del conocimiento y experiencia de desarrolladores individuales.
- *Falta de mecanismos de reflexión y reutilización en el ámbito de información, experiencias de diseño y estructura física de la aplicación web* (Nanard *et al.*, 1998; Rossi *et al.*, 1997; UIML, 2000), lo cual dificulta la transmisión de conocimiento y la generalización de diseños efectivos que mejoren su calidad global. Esta falta de reutilización crea entornos poco predecibles y sin

consistencia que aumentan la desorientación del usuario al navegar por los mismos.

- *Falta de mecanismos de evaluación del proceso de generación de la aplicación* (Lowe *et al.*, 1999): la no utilización de una metodología priva a las aplicaciones hipermediales de una base valiosa para evaluar las fases del proceso de desarrollo o medir atributos tan importantes como la facilidad de uso (accesibilidad, orientación, control de usuario) o el coste.

- *Falta de un proceso sistemático de verificación* de aplicaciones. Dado que la tolerancia del usuario a errores en aplicaciones hipermediales es muy bajo (Isakowitz *et al.*, 1995), el proceso de verificación debe ser aún más exhaustivo y sistemático que en los proyectos tradicionales de desarrollo de software. Sin embargo, esta tarea es mucho más difícil sin una metodología que sustente dicho proceso de desarrollo.

- *Necesidad de ejecución manual de tareas repetitivas*, susceptibles de introducir errores en la aplicación. Aún en la actualidad nos encontramos con desarrollos en los que el implementador se ve obligado a realizar actividades en las que se tiene una alta probabilidad de cometer errores (uso de nombres de campos coincidentes con los existentes en la base de datos, manejo de manera explícita el enlazado de las páginas, programación manual de los controles de entradas de usuario, etc.).

- *Dificultad de versionado, prototipado y personalización* de aplicaciones (Fernández *et al.*, 1999). La falta de un modelo conceptual obliga a la duplicación de ficheros y a su posterior modificación manual, lo cual causa errores e inconsistencias difíciles de detectar sin una comprobación exhaustiva, al mismo tiempo que ralentiza el proceso de desarrollo de aplicaciones. Este coste es mucho mayor debido a que el ciclo de vida de estos productos software es mucho más corto que el de las aplicaciones tradicionales.

- *Dificultad de migración* a nuevos entornos de desarrollo, como son los dispositivos móviles, dispositivos de voz, etc. (UIML, 2000). Los procesos de desarrollo artesanales suelen incluir asun-

ciones desde sus primeras fases acerca de plataformas de desarrollo, tecnologías utilizadas y dispositivos de acceso, lo cual impide la reutilización de los artefactos resultantes ante cambios en el entorno, que son la tónica general en los ambientes web.

- *Falta de mecanismos efectivos de implementación de ayuda contextual.* Tradicionalmente, la documentación de aplicaciones construidas mediante procesos de desarrollo *ad hoc* se limita (en el mejor de los casos) a la extracción de una descripción detallada de las características de la aplicación una vez implementada, y a su posterior enlazado con las correspondientes pantallas de interfaz. Este tipo de ayuda es insensible al contexto (tipo de usuario, condiciones de ejecución, etc.). Aún más, un cambio en la interfaz de usuario supone tener que modificar manualmente la ayuda. Sin embargo, la indefinición intrínseca del entorno de uso actual de las aplicaciones web sugiere la necesidad de inclusión de otro tipo de explicaciones, relativas a aspectos como por qué está deshabilitado un comando, secuencias de acciones permitidas sobre un objeto, etc. Este tipo de ayuda contextual resulta imposible de proporcionar sin una sincronización entre las actividades de desarrollo de la aplicación y las actividades de elaboración de la documentación (Molina, 1998).
- *Limitación del tipo y uso de herramientas* (tradicionalmente circunscritas a editores de páginas físicas y/o elementos multimedia) que soporten el desarrollo de la aplicación y agilicen su construcción.

Por otra parte, *de cara al usuario final*, todas estas dificultades de desarrollo se traducen en aplicaciones más difíciles de manejar, y la falta de un esquema conceptual subyacente ha demostrado ser la causa de problemas tan comunes en las aplicaciones actuales como (Nanard *et al.*, 1998; Troyer & Leune, 1998):

- *Carencias en la organización de datos*: poco intuitiva, con procesos de navegación mal estructurados o guiados, que incrementan la dificultad para que el usuario cree un esquema mental de

la información y le obligan a navegar de manera ineficiente e ineficaz por los sitios web (Atzeni *et al.*, 1998), debido a su aparente inconexión (falta de coherencia) o dificultad de emplazamiento y localización dentro de su estructura. Aunque a nivel de coherencia local (de contenido) parece lógico confiar en la habilidad de escritura del generador de contenido, a nivel de coherencia global es importante limitar la fragmentación arbitraria característica del hipertexto, y eso se puede conseguir con un buen esquema conceptual que indique por dónde es más lógico dividir la presentación de información, y cómo relacionar dicha información.

- *Carencias en la información*: enlaces rotos, información obsoleta, inconexa, incompleta, inconsistente o incluso inexistente. Todas estas carencias aumentan la desconfianza del usuario ante la aplicación y dificultan por tanto su uso.
- *Dificultad de identificación de un objetivo* claro del sitio web.
- *Diseño descuidado*: páginas demasiado pesadas, con fuentes o colores poco apropiados.

Todo lo mencionado anteriormente ha provocado que en los últimos años se haya investigado de manera exhaustiva sobre los conceptos y procesos que deberían intervenir en el desarrollo efectivo de estas aplicaciones para paliar en lo posible todos estos problemas. A continuación se presenta una breve descripción de las principales aproximaciones presentadas como resultado del trabajo de distintos grupos de investigación, y se hace especial hincapié en aquéllas que, debido a su concepción y enfoque, resultan más relevantes desde el punto de vista de este trabajo.

2.2 Propuestas de procesos, modelos de referencia y modelos de producto en aplicaciones web

Con el fin de resolver los problemas detectados en la sección anterior, distintos autores han adoptado diversos enfoques, entre los que destacamos:

- Sistematización del proceso de desarrollo
- Sistematización de los constructores hipermediales y su semántica
- Sistematización de las actividades específicas de las aplicaciones hipermediales

En primer lugar, existen aproximaciones centradas en el estudio exhaustivo no de modelos de diseño de aplicación (aunque puedan esbozar algunas guías al respecto) sino sobre todo de modelos de *proceso* que incrementen la calidad de las aplicaciones web. Entre ellas destacamos OO-Pattern (Thomson *et al.*, 1998), HFPM (Hypermedia Flexible Process Model (Olsina, 1998)) y el modelo de referencia para el proceso de desarrollo hipermedial de Lowe y Hall (Lowe & Hall, 1999). Este tipo de aproximaciones dan una solución general al problema de modelado, y normalmente no incluyen una descripción detallada de los artefactos que deben surgir como resultado de aplicar las actividades que proponen.

Por otro lado existen numerosas propuestas de formalización de marcos de referencia, cuyo principal objetivo es identificar el conjunto de abstracciones relevantes en el ámbito de las aplicaciones hipermediales, homogeneizar el vocabulario utilizado y facilitar la comparación de propuestas y la validación de modelos. De entre ellos destacan el Modelo de Referencia Hipertextual DEXTER (Halasz & Schwartz, 1994), que establece la separación entre contenido, estructura de navegación y presentación, el Modelo Hipermedial de Amsterdam (Hardman *et al.*, 1994), que añade como contribución más importante al modelo DEXTER la noción de tiempo, la familia de modelos hipermediales DORTMUND (Toch-

termann & Dittrich, 1996), que define de forma explícita los distintos tipos de datos soportados por las aplicaciones hipermediales, el Modelo de Aplicación Hipermedial Adaptiva AHAM (Bra *et al.*, 1999), que es el primer modelo de referencia centrado en aplicaciones personalizadas, y el Modelo de Referencia de Munich para Sistemas Hipermediales Adaptivos (Koch, 2001) que amplía el AHAM con una especificación más detallada del modelo de usuario y modelo de adaptación.

Por último, numerosas propuestas se han centrado en definir las actividades y modelos necesarios para reflejar la idiosincrasia de las aplicaciones web. Es precisamente en este último aspecto en el que se centra OO-H, por lo que a continuación daremos una visión general de las más relevantes para facilitar así su posterior comparación con nuestra aproximación.

2.3 Principales aproximaciones al Modelado Hipermedial

El interés dentro de la comunidad científica por los métodos y metodologías de modelado hipermedial ha sufrido un crecimiento exponencial desde que a principios de los 90 se presentó HDM (Garzotto *et al.*, 1993). Como primera aproximación al estudio de las distintas propuestas, y siguiendo el enfoque presentado en (Retschitzegger & Schwinger, 2000), las hemos categorizado en base a su nivel de sofisticación en tres *generaciones* (ver Fig. 2.1).

En general, tal y como se observa en la Fig. 2.1, las propuestas de modelado pueden ser divididas en dos grandes familias: aquéllas derivadas de modelos clásicos de datos (modelo Entidad-Relación) y aquéllas derivadas de modelos Orientados a Objetos (principalmente OMT y su sucesor UML).

La primera generación de métodos (aproximadamente hasta mediados de los años 90) sienta las bases de la Ingeniería Web al incluir por primera vez conceptos como constructores de navegación o promover la separación entre estructura de navegación y

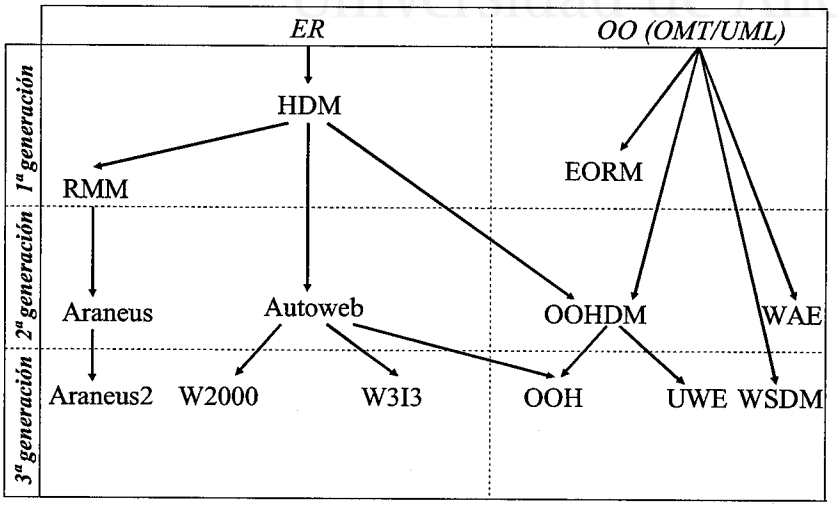

Figura 2.1. Generaciones de aproximaciones hipermediales

contenido durante el proceso de desarrollo. Como principales exponentes de estos métodos pioneros destacan por un lado HDM y RMM, ambos ejemplos de modelos dirigidos por datos, y por el otro EORM como ejemplo ilustrativo del impacto que las tecnologías orientadas al objeto ha tenido en la comunidad hipermedial desde sus orígenes.

La segunda generación de métodos (segunda mitad de los 90) refina los primeros modelos e incluye nuevos conceptos. Entre ellos destaca la introducción de soporte para funcionalidad básica (altas, bajas y modificaciones de datos) y los primeros esbozos de proceso, donde modelado conceptual, lógico y físico están por primera vez claramente delimitados. En esta segunda generación incluimos métodos como ADM, Autoweb, OOHDM o WAE.

La tercera generación de métodos (años 2000-2002) ha profundizado en este soporte para la funcionalidad. También se ha enfatizado el papel del usuario en los métodos, y se han producido avances hacia la estandarización de notaciones, procesos y lenguajes textuales de especificación. Destacan propuestas como ADM-2, WebML, W2000, UWE, WSDM o el propio OO-H, tal y como justificaremos a lo largo del capítulo.

A la hora de presentar el estudio de las aproximaciones más relevantes desde el punto de vista de este trabajo, nos hemos centrado en aquéllas que abordan el modelado de aplicaciones web dinámicas a partir de un repositorio de datos altamente estructurado (normalmente manejado por un Sistema de Gestión de Bases de Datos (SGBD)). Además, en el resto del capítulo nos centraremos en las tareas de análisis y diseño, que están en el corazón del presente trabajo, y que constituyen el núcleo central alrededor del cual giran la mayoría de las propuestas estudiadas.

2.3.1 HDM: Hypermedia Design Method

El Método de Diseño Hipermedial HDM (Garzotto *et al.*, 1991; Garzotto *et al.*, 1993) constituye una de las primeras propuestas de modelado hipermedial genérico (no restringido a la web). HDM parte de un modelo E-R extendido que refleja lo que HDM denomina *hiperbase* (modelo de dominio), y sobre el cual define un *modelo de acceso*. Otras características relevantes son la separación de estructura y contenido como modo de conseguir un desarrollo sistemático de la aplicación, una propuesta de taxonomía de enlaces y un esfuerzo explícito de automatización de algunos de sus pasos (e.g. generación automática de determinados tipos de enlace).

Los principales conceptos incluidos en esta propuesta son:

- Entidades: instancias de un tipo de entidad. Es la mínima unidad autónoma del sistema, y representa un objeto navegable.
- Jerarquía de Componentes: jerarquía de partes en las que se puede subdividir cada entidad.

- Unidades: componentes hoja con una perspectiva determinada.
- Enlaces: representan la semántica de navegación del sistema y pueden ser de aplicación (instancias de 'tipos de enlace', i.e. enlaces entre 'tipos de entidad'), de estructura (enlaces entre componentes de la misma entidad) o de perspectiva (enlaces entre unidades).
- *Outlines*: estructuras jerárquicas de acceso que enriquecen la navegación del esquema.

HDM dispone de un entorno CAWE llamado HSDL (Kesseler, 1995) que se basa en una versión enriquecida de HDM y permite la generación de aplicaciones web estáticas.

Por último, queremos destacar cómo HDM ha inspirado numerosos métodos, entre los que destacan OOHDM, RMM, HDMlite o WebML, tal y como veremos a continuación.

2.3.2 Autoweb

Autoweb (Fraternali & Paolini, 1998; Fraternali & Paolini, 2000) es una propuesta metodológica producida como evolución de HDM al entorno web, e incluye una notación de modelado conocida como HDM-lite. Como características fundamentales, Autoweb separa claramente los tres niveles de modelado: contenido, navegación y presentación abstracta (independiente del lenguaje de implementación final). Un soporte CAWE completo formado por un entorno visual para HDM-lite (VHDM) y un generador de páginas (Fraternali *et al.*, 1998), el reuso de esqueletos de aplicación y estilos de presentación y un conjunto de algoritmos de transformación entre modelos agiliza la construcción de aplicaciones. Además, su arquitectura tres capas aumenta la flexibilidad de las aplicaciones resultantes. Por último, destaca el conjunto de guías de diseño para aumentar la calidad de la interfaz generada.

HDM-lite conserva el concepto introducido en HDM de entidad como agregación de componentes. También distingue entre enlaces semánticos (tipo L) y enlaces estructurales (tipo P) e incluye un

esquema de acceso formado por travesías y colecciones, sobre las que se definen *modos de navegación*: índice, visita guiada, visita guiada indizada o mostrar todo. Estos modos de navegación han tenido una gran aceptación y actualmente están presentes en la mayoría de métodos y metodologías existentes.

2.3.3 W2000

Finalmente, HDM es también un antecesor directo de la metodología W2000 (Baresi *et al.*, 2001), que define un marco para el diseño de aplicaciones web mediante la combinación de elementos de HDM con elementos propuestos en el seno de UML, prestando especial atención al conjunto de dependencias entre las distintas tareas que lo componen. El lenguaje de modelado resultante de esta combinación de HDM y UML recibe el nombre de HDM2000. La principal novedad de esta aproximación es el reconocimiento del papel primordial que juegan los aspectos funcionales y evolutivos en las aplicaciones web. Destaca, por un lado, la separación explícita entre requisitos funcionales y requisitos navegacionales, a los que se dota de un tratamiento diferenciado. Además, se consideran de manera explícita los aspectos temporales mediante el uso de diagramas de estados UML.

2.3.4 W3I3

W3I3 (Ceri *et al.*, 2000) es una de las propuestas más estables existentes en la actualidad. Incluye un proceso, una notación y un lenguaje para el desarrollo de aplicaciones web donde el perfilado de los usuarios, la personalización y el comportamiento reactivo (mediante reglas de negocio) son factores clave. Su proceso de modelado resalta las perspectivas de estructura, derivación, navegación, composición de páginas y presentación, y propone modelos que reflejan cada una de estas perspectivas. Destaca su notación extremadamente intuitiva, así como el hecho de que ha sido la primera propuesta que ha proporcionado un entorno CAWE totalmente funcional que, además de incluir un entorno visual para

la especificación de los esquemas conceptuales de la aplicación, es capaz de almacenar dicha especificación visual mediante un conjunto de documentos XML y, mediante un conjunto de herramientas comerciales conocidas como WebRatio (WebRatio, 2002), generar aplicaciones web operativas.

Como características particulares, WebML incluye un lenguaje de especificación de consultas llamado WebML-OQL que le permite enriquecer el esquema estructural con información redundante. Soportado por un lenguaje de especificación con sintaxis XML conocido como WebML, considera de manera explícita la especificación de operaciones de mantenimiento de datos y, más recientemente, la inclusión de patrones que inciden en métricas contrastadas de calidad de aplicaciones web (Fraternali *et al.*, 2002).

2.3.5 RMM: Relationship Management Methodology

Otra metodología pionera en el campo del modelado de aplicaciones web es la Metodología para el Manejo de Relaciones (RMM (Isakowitz *et al.*, 1995)). Esta aproximación parte de un modelo ER que se enriquece con un conjunto de primitivas de navegación, dando lugar a un modelo denominado RMDM[1].

Como características más relevantes, RMM es la primera aproximación (hasta lo que alcanza nuestro conocimiento) que propone un proceso sistemático para el diseño y la construcción de aplicaciones hipermediales. En primer lugar, durante la tarea de diseño E-R, se identifican entidades y relaciones, que se convertirán en nodos y enlaces en la aplicación hipermedial resultante. En el segundo paso (diseño de particiones[2]), se agrupan los atributos de entidad para su presentación, y se definen nuevas estructuras de navegación para la conexión de dichas particiones. Estas particiones pueden ser vistas como unidades de presentación y por tanto transformables en páginas en la aplicación web resultante. En un tercer paso RMM especifica la navegación mediante primitivas de

[1] *Relationship Management Data Model*
[2] *slices*

acceso ya conocidas, como enlaces, grupos (menús), índices y visitas guiadas. A partir de ahí, un diseño de presentación precede a un conjunto de tareas orientadas a la implementación y testeo de la aplicación modelada. Además, RMM separa claramente el análisis de domino (modelo ER) del modelo de navegación y de la agrupación de información para su visualización. La herramienta CASE RMC (Díaz *et al.*, 1995) permite la generación automática de código HTML (páginas estáticas) a partir de los modelos RMDM introducidos.

2.3.6 Araneus

Basada en RMM, otra propuesta relevante orientada a la construcción de aplicaciones web es Araneus (Atzeni *et al.*, 1997; Atzeni *et al.*, 1998). Araneus incluye, al igual que la mayoría de las propuestas presentadas hasta el momento, un proceso de diseño centrado en la autoría de la aplicación, y lo complementa con una notación propia.

Versiones recientes del método (Mecca *et al.*, 1999a; Mecca *et al.*, 1999b; Atzeni & Parente, 2001) incluyen manejo de usuarios, control de flujo de la aplicación y operaciones (transaccionales o no) definidas como acciones asociadas con páginas o enlaces. Así, para la construcción de la aplicación, Araneus parte de un diagrama de actividad que especifica el flujo de la aplicación. Para cada etapa se indica el conjunto de casos de uso que la integran y, para cada caso de uso, se provee una especificación lógica de las páginas de aplicación que le dan respuesta, mediante el lenguaje de modelado de datos ADM (*Araneus Data Model*), cuya versión más reciente es ADM-2 y cuyo principal constructor es el esquema de página.

Paralelamente, a partir del ER y de los casos de uso se define un modelo conceptual de navegación (NCM, *Navigation Conceptual Model*) similar al que presenta RMM, donde se detectan los conceptos clave y los caminos y estructuras de acceso relevantes para la aplicación en su conjunto. El NCM sirve de base para la integración de los distintos ADM (principal aportación respecto a RMM), y este modelo integrado se enriquece con la especificación

lógica de la base de datos mediante el lenguaje de descripción textual PENELOPE. Esta especificación se complementa con un conjunto de plantillas en HTML que contienen el diseño de la presentación y permiten generar la aplicación para la plataforma deseada. Para ello Araneus incluye un entorno de generación, denominado Homer (Mecca *et al.*, 1999b), que da soporte a la metodología.

Resulta interesante constatar cómo Araneus define el primer lenguaje de especificación textual (hasta lo que alcanza nuestro conocimiento) aplicado a la descripción de aplicaciones web, adelantándose a la tan común especificación XML y XSL incluida en muchos métodos de desarrollo actuales.

2.3.7 EORM: Enhanced Object-Oriented Relationship Methodology

Paralelamente a los modelos y metodologías basados en el paradigma relacional, surge EORM (Lange, 1993; Lange, 1996), basado en el paradigma objetual. EORM incluye un proceso iterativo que, centrado en la fase de autoría de la aplicación, propone el enriquecimiento de un modelo orientado al objeto mediante la representación de relaciones entre objetos (enlaces o links) también en forma de objetos. EORM argumenta esta aproximación apuntando como principales ventajas el enriquecimiento semántico de las relaciones (debido a que se convierten en constructores extensibles), la posibilidad de que estas relaciones participen en otras relaciones o posibilidad de inclusión de distintos tipos de enlace (con un tipo de funcionalidad asociada) como parte integrante de librerías reutilizables. EORM viene acompañada de una herramienta CAWE, ODMTool, que, junto a un generador comercial de Interfaces Gráficas de Usuario denominado ONTOS Studio y un Sistema de Gestión de Base de Datos Orientado a Objetos, permite el diseño interactivo de esquemas EORM y la generación de código fuente, inicialmente en C++, de las clases incluidas en estos esquemas.

2.3.8 MacWeb

Otra propuesta orientada al objeto es MacWeb (Nanard & Nanard, 1995). Centrada en la autoría de la interfaz de usuario de la aplicación, incluye un entorno que integra un proceso de desarrollo clásico (contenido, navegación, presentación, implementación y pruebas). Como principales novedades considera de manera explícita el proceso mental que sigue el diseñador a la hora de aplicar los pasos de un método, lo cual les lleva a afirmar que el proceso de desarrollo de una aplicación debe poder realizarse tanto con una aproximación top-down como bottom-up. Para conseguirlo, proponen un soporte que permite por un lado la *instanciación* a partir de modelos genéricos (e.g. diagrama de clases) mediante el uso de técnicas de prototipado ligero, y por otro la *conceptualización* a partir de modelos concretos (e.g. a partir de diagramas de objetos y el uso de técnicas de aprendizaje estructural). La implicación más novedosa de este hecho es la consideración explícita del usuario en el desarrollo de la aplicación y la separación explícita entre interfaz de usuario (que incluye contenido, navegación y presentación) y la funcionalidad interna de la aplicación.

2.3.9 OOHDM: Object-Oriented Hypermedia Design Method

También orientada al objeto y también heredera de conceptos planteados en HDM, OOHDM (Schwabe *et al.*, 1996; Schwabe & Almeida, 1998; Schwabe *et al.*, 1999a)) considera que una aplicación es una vista sobre un modelo conceptual. OOHDM utiliza una mezcla de notación estándar (en la actualidad UML) y notación propietaria, e incluye un proceso de autoría de la aplicación que comprende cuatro actividades:

- Fase de Análisis de Dominio: incluye la elaboración de un modelo de negocio (diagrama de clases UML) y, en versiones más recientes (Schwabe *et al.*, 2001), también un modelo de requisitos de usuario (diagrama de casos de uso UML) y un modelo

que, con una notación propietaria, refleja el modo de interacción usuario-sistema (diagrama de interacción de usuario).

- Fase de diseño de navegación: comprende el diseño de los nodos de navegación, que se constituyen como vistas sobre el modelo de dominio, y un diagrama de contexto de navegación, construido a partir del diagrama de interacción de usuario. El diagrama de contexto de navegación está formado por clases de contexto y estructuras de navegación. Las clases de contexto incluyen tanto restricciones sobre la población de la clase como aspectos relacionados con la ordenación, permisos, etc. Sobre estas clases de contexto se pueden definir además vistas de rol. Las estructuras de navegación por otro lado enriquecen este diagrama con patrones de navegación conocidos (menús, índices, etc.) que permiten navegar a través de las clases de contexto.
- Fase de diseño de interfaz abstracta: en esta fase se especifican, entre otros aspectos, la apariencia de los objetos de navegación, así como los objetos activadores de esa navegación y las distintas transformaciones que puede sufrir la interfaz. Para ello se utilizan ADV (Abstract Data Views).
- Fase de implementación: la implementación de los ADV obtenidos en la fase anterior se realizan mediante una aproximación de plantillas especificadas en el lenguaje de script Lua (LUA, 2002).

Como principales aportaciones de OOHDM destacamos su orientación al objeto, en la que fue un método pionero junto con EORM, y la separación explícita de distintos aspectos. Por un lado se diferencian por primera vez clases de dominio y clases de navegación (nodos), reconociendo así la diferencia entre ambos conceptos. Por otro, estos nodos son el punto de partida para distintos contextos de navegación. La especificación de una interfaz abstracta, independiente de dispositivo, es otro de sus aspectos más representativos. En versiones recientes del método también se ha empezado a prestar atención a la funcionalidad, se ha pro-

puesto una traducción de los modelos a RDF[3] y se ha adoptado un enfoque claramente dirigido por los requisitos de usuario.

OOHDM incluye OOHDM-Web (Schwabe *et al.*, 1999b) como entorno de desarrollo. OOHDM-Web permite el prototipado rápido de aplicaciones, al proporcionar un mapeo directo entre los constructores de navegación e interfaz de OOHDM y una librería de funciones en el entorno de *scripting* CGI-LUA. De este modo se generan aplicaciones hipermediales basadas en módulos CGI que producen páginas dinámicas, cuyo contenido se obtiene de una base de datos y se integra con el conjunto de plantillas predefinidas en el entorno.

2.3.10 WSDM: Web Site Design Method

Otra aproximación relevante es el Método de Diseño de Lugares Web (WSDM (Troyer & Leune, 1998)). Como principal novedad, este método, fiel al paradigma orientado a objetos, se aleja de las aproximaciones dirigidas por los datos y se plantea un proceso dirigido por los requisitos de la audiencia.

Para conseguir su objetivo, WSDM propone cinco fases: especificación de la misión de la aplicación (y audiencia objetivo), modelado de la audiencia, diseño conceptual, diseño de implementación e implementación.

Durante el modelado de la audiencia se clasifica y se caracteriza a los usuarios. El resultado es un modelo jerarquizado de usuarios con sus características y necesidades.

Durante la fase de diseño conceptual se modela tanto el espacio de información como la funcionalidad y la navegación requerida por cada tipo de usuario. El modelo conceptual final se consigue mediante la integración de los nodos de información y de los nodos funcionales en el modelo de navegación de la aplicación web.

La fase de diseño de implementación se centra en la estructura y apariencia las páginas que conforman la aplicación web. La es-

[3] Resource Description Format (RDF, 2002)

tructura de página se deriva del modelo de navegación. Además, si la aplicación tiene soporte (total o parcial) de una Base de Datos (BD), durante esta fase se diseña también el esquema lógico de dicha BD (posiblemente a partir del modelo de información de negocio).

Por último, en la fase de implementación se materializan los modelos para el entorno de ejecución elegido. WSDM incluye un entorno CAWE para la generación semiautomática de la aplicación.

Este método define su propia notación gráfica, así como un conjunto de palabras reservadas para los objetos del modelo navegacional. También destaca la separación entre requisitos informacionales, funcionales y navegacionales .

2.3.11 SOHDM: Scenario-based Object-oriented Hypermedia Design Methodology

La Metodología de Diseño Hipermedial Orientada al Objeto y Basada en Escenarios (Lee *et al.*, 1998) hereda conceptos de otras metodologías como RMM, OOHDM o EORM y propone un proceso de desarrollo dividido en seis fases: análisis de dominio, modelado de objetos, diseño de vistas, diseño de navegación, diseño de implementación y construcción. La principal novedad de este método es el uso de escenarios, descritos mediante el uso de diagramas de actividad donde intervienen eventos, actividades y primitivas de modelado relacionadas con los flujos.

A nivel de navegación SOHDM distingue entre primitivas de acceso (nodos de acceso a la estructura) y vistas de objeto (nodos de información) que se categorizan en función de su modo de derivación a partir del modelo de negocio. Además, esta propuesta incluye su propia notación.

2.3.12 WebArchitect

WebArchitect (Takahashi & Ling, 1997) es otra propuesta que se centra en la arquitectura y las funciones de los lugares web, más

que en la apariencia de cada recurso web (página). A nivel de análisis, WebArchitect combina el uso de un modelo E-R para el modelado de los aspectos estáticos de los sistemas basados en web con el uso de escenarios para modelar sus aspectos dinámicos. El análisis y diseño E-R se basa en RMM y define las entidades, sus roles (agente, producto, evento) y las relaciones existentes entre ellas. El análisis de escenarios por su parte define cómo se acceden los recursos web, cómo se usan y por quién. El método también define atributos de cada recurso web, que son utilizados para el mantenimiento del recurso. La implementación y mantenimiento de la aplicación resultante es soportada por una herramienta del mismo nombre, que permite a los diseñadores manipular de forma directa meta-enlaces entre recursos web organizados en un árbol jerárquico. Por otro lado, la visualización de las aplicaciones resultantes se realiza mediante un cliente web denominado Pilot-Boat, que navega y deja que los usuarios colaboren a través de los lugares web.

2.3.13 UWE: UML-based Web Engineering

La propuesta de Ingeniería Web basada en UML (UWE (Koch, 2000)) es la primera propuesta, hasta lo que alcanza nuestro conocimiento, que aúna una metodología detallada para el proceso de autoría de aplicaciones con una definición exhaustiva del proceso de diseño que debe ser utilizado. Este proceso, iterativo e incremental, incluye flujos de trabajo y puntos de control, y sus fases coinciden con las propuestas en el Proceso Unificado de Modelado (UP (Jacobson *et al.*, 1999)): incepción, elaboración, construcción, transición y mantenimiento.

UWE está especializada en la especificación de aplicaciones adaptivas, y por tanto hace especial hincapié en características de personalización, como es la definición de un modelo de usuario o una etapa de definición de características adaptivas de la navegación en función de las preferencias, conocimiento o tareas de usuario.

Por lo que respecta al proceso de autoría de la aplicación, UWE hace uso exclusivo de estándares reconocidos como UML y el

lenguaje de especificación de restricciones asociado OCL[4]. Para simplificar la captura de la idiosincrasia de las aplicaciones web, UWE propone una extensión (perfil, si utilizamos la nomenclatura UML) que se utiliza a lo largo del proceso de autoría. Este proceso de autoría está dividido en cuatro pasos o actividades: análisis de requisitos (reflejado en un modelo de casos de uso), diseño conceptual (materializado en un modelo de dominio), diseño navegacional (origen de dos modelos: el modelo de espacio de navegación y modelo de estructura de navegación) y diseño de presentación (cuyo flujo se determina mediante modelos estándares de interacción UML). Una extensión de ArgoUML (ArgoUML, 2002) conocida como ArgoUWE, soporta la aproximación.

Otras características relevantes del proceso y método de autoría de UWE son el uso del paradigma objetual, su orientación al usuario, la definición de un metamodelo (modelo de referencia) que da soporte al método y el grado de formalismo que alcanza debido al soporte que proporciona para la definición de restricciones sobre los modelos.

2.3.14 WAE: UML Extension for building Web Applications

De manera similar a UWE, pero sin tener en cuenta abstracciones y constructores bien conocidos en el seno de la comunidad hipermedial, la extensión para aplicaciones web (WAE (Conallen, 1999)) presentada por Conallen define, mediante el uso exclusivo de los modelos existentes en UML, la semántica contenida en una aplicación web. Conallen considera que una aplicación web es una versión especializada de una aplicación cliente/servidor, donde la conexión entre ambos componentes sólo existe durante la petición de página. Para ello incluye estereotipos para componentes, clases, métodos y asociaciones. Estos estereotipos son apropiados para el modelado de aspectos relacionados con apariencia e implementación. Sorprende constatar como Conallen no considera la separación entre navegación y presentación, piedra angular de

[4] Object Constraint Language (UML 1.4, 2001)

la mayoría de las aproximaciones hipermediales estudiadas. Por el contrario, propone un conjunto de modelos de implementación que, en el contexto del Proceso Unificado de Rational (Rational Unified Process (RUP, 2001)), giran en torno a la arquitectura de la aplicación.

2.3.15 WebApp

El marco de referencia WebApp (Enguix & Davis, 1999) propone un entorno multi-paradigma que incluye componentes de diversos tipos: procedurales, declarativos, basados en objetos, OO, dirigidos por evento, etc. Al mismo tiempo, propone una aproximación metodológica semiformal que permite la combinación de modelos hipermediales y modelos tradicionales de la Ingeniería del Software. Por último, incluye un conjunto de modelos y notación capaces de representar la interacción de aplicaciones web cliente-servidor desde una perspectiva de página, coincidiendo así con aproximaciones como la de Conallen.

WebApp propone un proceso de desarrollo prescriptivo de seis fases: recogida de requisitos, modelado de simulación, modelado de arquitectura, análisis de escenarios de implementación, modelado formal y modelado de la implementación, cuyo resultado final es la aplicación implementada. Es precisamente en los modelos de implementación donde conceptos como marcos, formularios, componentes activables de cliente, eventos, parámetros, etc. son constructores básicos.

2.3.16 WebComposition

Con un enfoque distinto al de las aproximaciones presentadas hasta el momento, WebComposition (Gaedke *et al.*, 1999; Gaedke & Rehse, 2000; Gellersen & Gaedke, 1999) es un modelo Orientado al Objeto y Basado en Componentes que actúa de puente entre los modelos de diseño y los modelos de implementación. WebComposition parte de un conjunto de modelos de diseño, expresados en

OOHDM, y traduce sus abstracciones a un lenguaje denominado WCML[5] que, basado en XML, permite la definición de componentes a cualquier nivel de abstracción.

En esta aproximación las entidades web son vistas como objetos componentes, y por tanto dotadas de un estado y un comportamiento asociado a cada entidad. Los componentes pueden modelar entidades web de una complejidad arbitraria: enlace individual, una página completa, un fragmento de código interpretado, etc. Web Composition también modela los recursos basados en ficheros mediante componentes, lo cual permite el reuso y el mantenimiento, pero no aborda el modelado del comportamiento de las aplicaciones en entornos de desarrollo.

La principal aportación de WebComposition es por tanto la creación de una nueva línea de investigación dentro de la Ingeniería Web. Se trata de la *Ingeniería Web basada en componentes*[6], entendida como la producción eficiente en costes de aplicaciones web de alta calidad mediante el uso de un proceso definido que incluye el reuso sistemático de componentes y conocimiento de dominio. Esta línea sigue un enfoque orientado al servicio, que contrasta con los enfoques orientados a datos que hemos presentado en aproximaciones anteriores. Además, en ella la integración y el reuso son elementos claves. Con el fin de facilitar este reuso WebComposition incluye un repositorio extensible que permite la recuperación y clasificación de extensos conjuntos de componentes.

2.3.17 Aproximaciones basadas en motores de plantillas

Además de las aproximaciones presentadas hasta el momento, existen propuestas que presentan un menor nivel de abstracción, como son los motores de plantillas (*template engines*) que se centran en la especificación, normalmente mediante sintaxis XML, de modelos de implementación y estrategias de generación de código.

[5] WebComposition Markup Language
[6] Component-Based Web Engineering (CBWE)

Ejemplos de este tipo de propuestas son MyXML(MyXML, 2000) o UIML (UIML, 2000).

El objetivo principal del conjunto de herramientas de desarrollo conocido como MyXML es proporcionar una especificación de aplicación que, independiente de plataforma y lenguaje de implementación final, preserve la separación entre contenido, apariencia y lógica de negocio con el fin de facilitar la mantenibilidad y el reuso. Para ello el espacio de nombres definido por MyXML proporciona elementos de plantilla que soportan la generación dinámica de contenido en tiempo de ejecución e integran conceptos populares como el uso de parámetros CGI o el acceso a bases de datos desde documentos MyXML. Su integración con XSL para definir la apariencia de interfaz permite la inclusión de información de contexto en las reglas de formateo. Como característica añadida, el motor de generación que acompaña a la propuesta está preparado para optimizar la eficiencia de las páginas (estáticas o dinámicas) generadas.

UIML[7] (UIML, 2000) por su parte es un lenguaje basado en XML cuyo objetivo es permitir la creación de interfaces de usuario compatibles con cualquier metáfora (interfaz gráfica, interfaz de voz, etc.), cualquier dispositivo, cualquier lenguaje objetivo y cualquier sistema operativo que esté corriendo sobre el dispositivo. Para ello UIML define un modelo de implementación exhaustivo que incluye tres aspectos fundamentales: la presentación (tipo de constructor físico, localización, fuentes, colores, etc.), el modelo de interacción del usuario con la interfaz (reacción de los constructores ante eventos de interfaz) y la conexión con lógica de negocio (mediante un mecanismo de eventos). Además destaca su relativa sencillez, con aproximadamente una treintena de etiquetas.

Desde este trabajo consideramos que estas propuestas pueden complementar los modelos de análisis y diseño comentados en la sección 2.3 y facilitar el proceso de acercamiento de las especificaciones de diseño al espacio de la implementación, simplificando así la generación de código desde aproximaciones conceptuales.

[7] User Interface Modeling Language

2.3.18 Aproximaciones que integran información semiestructurada

Existe otro conjunto de propuestas que afrontan la dicotomía de la información (estructurada y desestructurada) que interviene en la configuración de muchas aplicaciones hipermediales. Entre ellas destacamos W3DT (Bichler & Nusser, 1996) y Strudel (Fernández *et al.*, 1999).

W3DT (World Wide Web Design Technique) es una propuesta semiformal centrada en el espacio de la solución, y que por tanto integra como constructores conceptos como sitio web, página, índice, menú, formulario, enlace estático (enlace con páginas físicas preexistentes) o enlace dinámico (enlace con plantillas de página que representan páginas generadas en tiempo de ejecución). Ofrece un entorno de prototipado denominado W3DT WebDesigner que soporta la notación gráfica de la metodología y es capaz de generar un marco de páginas dinámicas en cada paso del proceso de diseño.

Por otro lado, mucho más evolucionada, Strudel es una aproximación declarativa para el tratamiento de información semiestructurada, normalmente basadas en grafos. Esta aproximación incluye lenguajes de consulta para la definición de la estructura de la aplicación. En Strudel, del mismo modo que en MyXML, la estructura de presentación se consigue mediante la mezcla de HTML (W3C, 2000) con un lenguaje de plantillas específico que es capaz de acceder las estructuras de información.

Nuevamente el tratamiento de información semiestructurada es un aspecto que complementa a las aproximaciones conceptuales más relevantes de la actualidad, y su integración en estas propuestas incrementaría su potencia expresiva.

2.3.19 Otras aproximaciones

Además de los enfoques presentados hasta el momento, existen aproximaciones de un mayor nivel de abstracción, pero centradas en el tratamiento de documentos de carácter eminentemente

estático, como es el caso de IDM (Intranet Design Methodology) (Lee, 1998). Esta propuesta propone una notación para especificar plantillas de documentos jerárquicas, y facilitar así su navegación y mantenimiento.

Por último, otras propuestas hacen uso exclusivo de mecanismos proporcionados por los SGBD. Así, OSM (Liddle, 2001) presenta un conjunto de modelos de implementación que incluyen el esquema lógico de la base de datos y donde se explicita, también en forma de tablas, las estructuras de información necesarias para implementar disparadores que reaccionen ante eventos del sistema.

Nuevamente estas perspectivas pueden ayudar al enriquecimiento de las aproximaciones hipermediales presentadas en la sección 2.3 y aumentar su flexibilidad y nivel de expresividad.

2.4 Comparativa de las distintas aproximaciones

La relativa falta de madurez de la Ingeniería Web, unida a la falta de consenso en cuanto a qué aspectos y a qué nivel de abstracción deben intervenir en un proceso de modelado, hacen difícil la tarea de contrastar las distintas aproximaciones. A pesar de ello en los últimos años se han planteado una serie de marcos de comparación que resumen de una manera muy ilustrativa las principales características de cada método. Entre ellos destacamos los trabajos de (Florescu *et al.*, 1998), (Fraternali, 1999), (Retschitzegger & Schwinger, 2000) y (Koch, 2001). Esta comparativa pretende complementarlas con nuevos aspectos, como es el grado de soporte a la funcionalidad, personalización o flujo de trabajo, todos ellos nuevos requisitos que deben ser cubiertos por los métodos de desarrollo hipermedial para dar respuesta a las necesidades actuales de los desarrolladores. Es importante destacar que esta comparativa se centra en evaluar los aspectos que inciden sobre el diseño de la interfaz. Esto quiere decir que cuando hablemos por ejemplo de soporte al modelado de eventos, nos referiremos a

si la aproximación proporciona mecanismos para comunicar *a la interfaz* que se ha producido un evento y poder así ésta cambiar su estado en consecuencia, independientemente de que a nivel de lógica de negocio se establezcan o no mecanismos de comunicación mediante eventos entre los distintos módulos que componen la aplicación.

Basándonos en el trabajo de documentación efectuado para la realización del presente trabajo podemos afirmar que, pese a lo que podría parecer en un principio, existen numerosas similitudes entre las distintas propuestas de modelado hipermedial, sobre todo entre aquéllas (las más numerosas) que abordan el modelado de aplicaciones dinámicas, intensivas en datos, donde la estructuración, la modularidad y la facilidad de uso son factores clave.

En general, estas propuestas identifican la necesidad de modelar la navegación de manera independiente a cualquier otro tipo de funcionalidad de la aplicación, aspecto que es obviado en métodos y metodologías estándares (e.g. UML, OMT, etc.). Dentro de esta navegación, un rasgo común es la definición de estructuras de navegación como vistas externas sobre estructuras de dominio. En este punto destaca el nivel de abstracción introducido, que es mayor que el presentado en los clásicos *siteviews* del sistema, simplificando de este modo los diagramas resultantes. Además, la mayoría de métodos existentes proporcionan un mecanismo de agrupación (explícito o implícito) de elementos relacionados para la cobertura de cada requisito de usuario. Por último, en este modelo es común distinguir entre estructuras de acceso y estructuras de información, y conceptos como nodo y enlace pertenecen al vocabulario estándar.

Otro rasgo común es la separación explícita de tres *niveles* de modelado: contenido, navegación y presentación (Retschitzegger & Schwinger, 2000), que dan lugar a distintos modelos. Las actividades de análisis de requisitos, diseño conceptual del contenido, diseño navegacional y diseño de presentación (con ligeras variantes) son comunes a la mayoría de los métodos estudiados. También existe un acuerdo generalizado en la necesidad de un proceso explícito que guíe todo el ciclo de vida de una aplicación

web, o que al menos proporcione una guía exhaustiva durante su fase de autoría. Este proceso debe ser preciso y sistemático, y debe incluir tantos mecanismos de automatización entre actividades como sea posible con el fin de acelerar el desarrollo de este tipo de aplicaciones.

Además, todas las propuestas estudiadas reconocen en mayor o menor medida la necesidad de una herramienta CAWE que dé soporte a todas las actividades planteadas[8].

Por último, la mayoría de las propuestas estudiadas aportan algún tipo de notación específica (bien propietaria, bien obtenida mediante mecanismos de extensión de lenguajes estándares) para plasmar de una manera sencilla la semántica de los conceptos particulares de la web, y reconocen la conveniencia de proporcionar mecanismos de especificación de restricciones para aumentar la precisión de los modelos. En este sentido, conceptos como nodo, colección, enlace o patrón de navegación son factores comunes en numerosas propuestas.

Desde este trabajo defendemos que, lejos de suponer un problema, esta compartición de conceptos es una de las razones que explican la rápida evolución de la Ingeniería Web. Experiencias recientes (Pastor, 2001; Schwabe *et al.*, 2002) demuestran, desde nuestro punto de vista, la confluencia de premisas que sustentan a los distintos métodos de diseño hipermedial, como es su orientación al usuario o la creciente importancia de la funcionalidad. Desde OO-H pensamos que esta suma de esfuerzos es necesaria para dotar a las empresas con el tipo de método y entorno de desarrollo que éstas demandan en un período de tiempo razonable.

Por otro lado, existe otro amplio rango de características que difieren entre las distintas aproximaciones. En este punto es importante destacar que la falta de soporte o el distinto nivel de detalle con el que se determinan ciertos rasgos en cada una de las propuestas, tal y como veremos a continuación, no debe ser

[8] Es conveniente hacer notar sin embargo que, aunque todas las propuestas afirman disponer estas herramientas, hasta lo que alcanza nuestro conocimiento sólo W3I3 (con una herramienta propietaria), WAE (con la edición de un perfil para Rational Rose) y nuestra aproximación han demostrado dicho soporte

vista como una crítica, sino como una prueba más de los distintos enfoques a partir de los cuales se abordan las distintas fases del proceso de desarrollo de una aplicación web. En efecto, como ya comentamos en la sección 1.4.1, el bagaje de la propuesta (hipermedial, relacional, objetual o dinámico) influye de manera decisiva en el proceso y modelos que intervienen en el desarrollo de la aplicación, y afectan de manera directa a aspectos como el grado de funcionalidad soportado (muy bajo en las propuestas que vienen del ámbito hipermedial, como HDM, y con una percepción totalmente orientada al servicio en aproximaciones dinámicas como WCML), la inclusión o no de lenguajes de especificación de restricciones y su uso para limitar tanto la información visualizada como los caminos disponibles a través de ella, el significado de los constructores utilizados (e.g. uso del término *enlace* para hacer referencia a relaciones abstractas o a conectores concretos entre páginas físicas) o el nivel de abstracción al que se diseñan los modelos.

A continuación comentaremos algunas de las divergencias más relevantes. Esta comparativa puede verse resumida en las Figs. 2.2 y 2.3.

CARACTERÍSTICAS GENERALES	WAE	W2000	Araneus	WSDM	OOHDM	W3I3	UWE	OO-H
Paradigma	OO	OO	Relacional	OO	OO	Relacional	OO	OO
Orientación	Usuario	Usuario	Datos	Usuario	Usuario	Datos	Usuario	Usuario
Ámbito de Aplicación	Aplicación completa	Aplicación completa	Aplicación Completa	Aplicación Completa	Aplicación Completa	Aplicación Completa	Aplicación Completa	Interfaz
Estándares	UML, RUP	UML	XML	UML	UML, RDF	XML	UML, UP	UML, XML
Modelo Referencia	Metamodelo UML	No	No	No	No	No	Modelo de Referencia de Munich	No
Proceso de Desarrollo	Ciclo Vida	Fase Autoría	Fase Autoría	Fase Autoría	Fase Autoría	Fase Autoría	Ciclo Vida	Fase Autoría

Figura 2.2. Rasgos generales de las principales aproximaciones

2.4.1 Paradigma utilizado

Cualquier aplicación, sea del tipo que sea, debe conocer el dominio en el que se mueve. Para ello todas las aproximaciones proporcionan un modelo de datos, que varía en función del tipo de entorno en que nos encontremos. En el caso de entornos de información al-

tamente estructurada, la tendencia más extendida es la adopción de una perspectiva relacional (modelo E-R), seguida en aproximaciones como HDM, RMM, WebML o ADM-2, o una perspectiva objetual (diagrama de clases), presente en aproximaciones como EORM, OOHDM o el propio OO-H.

2.4.2 Orientación

En cuanto al enfoque adoptado, podemos hablar de cuatro tendencias: métodos orientados al usuario, métodos dirigidos por datos, métodos orientados al servicio y métodos orientados a la tarea. Mientras los dos primeros son los predominantes en la actualidad (ver Fig. 2.2), creemos que las aproximaciones orientadas al servicio (tales como WebComposition) y aproximaciones orientadas a tareas (como es el caso de ConcurTaskTree (Paterno, 1999)) van a tener una gran influencia en la evolución de los métodos para incluir nuevos rasgos de la aplicación concernientes a su funcionalidad y flujo de trabajo.

2.4.3 Ámbito de Aplicación

Otro rasgo diferenciador es el objetivo de cada aproximación. Algunas propuestas (e.g. W2000 o WSDM) se ocupan del modelado de la aplicación completa. Otras propuestas (e.g. W3I3) van un paso más allá y generan una aplicación final operativa. El principal problema de este tipo de aproximaciones es que obvian muchos aspectos concernientes a la lógica de negocio de la aplicación. Es por este motivo que OO-H se ha separado de esta tendencia y ha delegado la tarea de modelado de lógica a métodos estándares que han probado su eficacia en este tipo de tareas, centrándose por tanto en el modelado y generación de la interfaz de la aplicación, y en su interconexión con módulos preexistentes.

2.4.4 Estándares Utilizados

La mayoría de aproximaciones estudiadas incluyen uno u otro tipo de estándares, bien sea a nivel de proceso, bien sea a nivel de modelos, notación o almacenamiento de la especificación. Hasta lo que alcanza nuestro conocimiento, sólo WAE y UWE extienden una notación estándar (UML) y utilizan un proceso también estandarizado (RUP y UP, respectivamente). El resto mezcla modelos estándares (e.g. Diagrama de Casos de Uso en OOHDM, WSDM u OO-H) con modelos basados en una notación propietaria. La especificación capturada mediante los distintos modelos es en algunos casos (W3I3, Araneus, OO-H) guardada como una especificación XML, mientras que en otro se confía en repositorios propietarios.

2.4.5 Uso de un modelo de referencia

Otro tema importante es el uso de un modelo de referencia. Los modelos de referencia formalizan los conceptos y relaciones que pueden ser capturados mediante un determinado método o metodología. Esta formalización es necesaria si se pretende realizar un análisis o una comparación rigurosa de los mismos, y aún más si se pretende avanzar hacia un estándar. Sin embargo, la relativa falta de madurez de los métodos existentes hace que sólo HDM (basado en el modelo de referencia de Dexter), WAE (basado en el metamodelo de UML) y UWE (basado en el modelo de referencia de Munich, que a su vez está basado en el modelo de referencia de Dexter) definan explícitamente un metamodelo de estas características.

2.4.6 Cobertura del proceso de desarrollo

Por último, la mayoría de las propuestas estudiadas abogan por un proceso iterativo e incremental, que facilite el desarrollo de versiones sucesivas de la aplicación en un período de tiempo razonable. La definición de este proceso sin embargo no abarca las

mismas actividades en todas ellas: mientras que propuestas como UWE o WAE son metodologías que abarcan todo el ciclo de vida de la aplicación (WAE extendiendo el RUP, y UWE incluyendo las actividades definidas en el Proceso Unificado), el resto se centran en la definición del subproceso que incluye las actividades de autoría de la aplicación, desde el modelado de requisitos funcionales hasta la validación final de la aplicación por parte del cliente, sin proporcionar guías detalladas para actividades como la planificación de iteraciones, el análisis de requisitos, mantenimiento de la aplicación, control de calidad, etc. A continuación veremos algunas diferencias significativas dentro de este proceso de autoría.

2.4.7 Proceso de Autoría

	WAE	W2000	ADM-2	WSDM	OOHDM	W3I3	UWE	OO-H
Proceso Autoría								
Modelado Independiente de Plataforma	☐	☑	☑	☑	☑	☑	☑	☑
Presencia de actividades de prototipado	☐	☐	☐	☐	☑	☑	☑	☑
Automatización de actividades	☐	☐	☐	☐	☑	☐	☑	☐
Soporte Flujo Trabajo	☐	☑	☑	☐	☐	☐	☐	☐
Funcionalidad								
Operaciones genéricas	☑	☐	☐	☐	☐	☑	☐	☑
Composición de operaciones	☐	☐	☐	☐	☐	☐	☐	☑
Soporte Modelado Transacciones	☐	☐	☑	☐	☐	☑	☐	☐
Soporte Modelado Eventos	☐	☐	☐	☐	☐	☑	☑	☑
Soporte Modelado Introducción Parámetros	☐	☐	☐	☐	☐	☐	☐	☑
Personalización								
Adaptación	☐	☑	☑	☑	☑	☑	☑	☑
Adaptividad	☐	☐	☐	☐	☑	☑	☑	☑
Proactividad	☐	☐	☐	☐	☐	☑	☑	☐
Soporte CAWE								
Prototipado	☐	☐	☐	☐	☐	☑	☐	☑
Soporte Guías Diseño	☐	☐	☐	☐	☐	☑	☐	☑
Generación automática código	☐	☐	☐	☐	☐	☑	☐	☑
Soporte para propagación del cambio	☑	☐	☐	☐	☐	☑	☑	☑

Figura 2.3. Rasgos específicos de las principales aproximaciones

Modelado Independiente de Plataforma. Una diferencia importante es la variación en el nivel de abstracción de las actividades incluidas en este proceso: aunque la mayoría de las propuestas (incluyendo OO-H) proporcionan una fase de análisis y diseño

conceptual de la aplicación, otras (e.g. WAE) trabajan directamente en el espacio de la solución, y sus constructores básicos son los recursos físicos (ficheros que representan páginas, estáticas o dinámicas, con una determinada tecnología asociada, o componentes con los que interactúan dichas páginas) y las relaciones que pueden establecerse entre dichos recursos.

Inclusión de Actividades de Prototipado. Por otro lado, algunas propuestas (e.g. OOHDM, UWE, W3I3, OO-H) formalizan actividades de prototipado, muy comunes en el ámbito del diseño de interfaces, como parte de la fase de ingeniería de la aplicación, con o sin apoyo de herramientas CAWE.

Automatización de actividades. Otra característica que diferencia a las distintas propuestas es el grado de automatización de actividades. Aunque potencialmente creemos que cualquiera de ellas puede automatizar el paso de una a otra actividad[9], sólo OOHDM y UWE proporcionan heurísticas explícitas para hacerlo.

Soporte Flujo Trabajo. Por último, otro tema en el que se está trabajando en la actualidad es el grado de soporte al flujo de trabajo de la aplicación. En general, la mayoría de las aproximaciones soportan la secuenciación de actividades para cada caso de uso de manera implícita, a través de sus modelos de navegación. Algunas propuestas (e.g. UWE) incluyen además diagramas de actividad que recogen secuencias de actividades para cada caso de uso de manera explícita. Sin embargo, no todas soportan el concepto de *estado*, que determina la accesibilidad en función de determinados criterios (tiempo, acceso previo a otras funciones, etc.) del usuario a unas u otras funcionalidades de la aplicación. En este sentido, Araneus parte de un diagrama de estados y una especificación declarativa que permite la definición de flujos de actividad. W2000 también incluye un diagrama de estados de estas características, que le permite además definir restricciones temporales sobre la accesibilidad a las distintas partes de la aplicación.

[9] De hecho ésta es una de las líneas de trabajo actuales en OO-H.

OO-HDM agrupa los Casos de Uso en función de estados de la aplicación, pero, hasta lo que alcanza nuestro conocimiento, esta información no se propaga al resto de fases del método.

2.4.8 Funcionalidad

A la hora de abordar la funcionalidad, existen dos corrientes principales, claramente diferenciadas: tratar a las operaciones como constructores independientes del método (unidades de WebML, constructores de OOHDM o palabras reservadas en WSDM) o, por el contrario, asociar la funcionalidad a los enlaces de navegación, como hace ADM-2 o el propio OO-H.

A continuación veremos el grado de soporte que distintos métodos proporcionan a distintos aspectos de esta funcionalidad.

Soporte a las operaciones genéricas. En general, los principales métodos de modelado hipermedial estudiados hasta el momento incluyen soporte para altas, bajas y modificaciones. Algunos métodos, provenientes del mundo de las BD, sólo proporcionan estas operaciones. Otros, provenientes del ámbito OO, permiten además capturar en su modelo de dominio cualquier otro tipo de operación genérica, que no obstante no se soporta en fases posteriores del método (WSDM, OOHDM). Por último, algunos métodos proporcionan constructores genéricos que permiten la inclusión de cualquier tipo de operación en los modelos (W3I3,OO-H y WAE, debido a que se trata de una extensión de UML).

Soporte composición de operaciones. Otro tema interesante, que será abordado en el capítulo 6, es el de la composición de operaciones y, de forma más evolucionada, la transaccionalidad de estas composiciones (lo que algunos autores llaman I-transacciones). Hasta lo que alcanza nuestro conocimiento, ningún método proporciona soporte al modelado transaccional de operaciones compuestas, y sólo OO-H ha realizado una primera propuesta de notación para abordar la composición de dichas operaciones.

Soporte Modelado Transacciones. Otro tema importante es el soporte a las transacciones. En este tema aproximaciones provenientes del mundo relacional (W3I3, ADM-2) han sido las primeras en incorporar mecanismos explícitos de modelado para ellas.

Soporte Modelado Eventos. Las aplicaciones no sólo tienen por qué reaccionar ante eventos de usuario, sino que también pueden hacerlo ante eventos del sistema o del entorno de la aplicación. Propuestas como UWE, WebML u OO-H proporcionan mecanismos[10] que le permite capturar determinados tipos de eventos y reaccionar en consecuencia.

Soporte al Modelado de Introducción de Parámetros. Si nos centramos en el modo de introducir parámetros en los servicios ofertados por la aplicación, la mayoría de los métodos confían en el tipo de dato para proporcionar uno u otro mecanismo de presentación que permita la introducción de los valores correspondientes. Sin embargo, en el caso de servicios arbitrarios este mecanismo no es suficiente. Hasta lo que alcanza nuestro conocimiento, sólo OO-H modela de manera explícita esta introducción de parámetros y proporciona una taxonomía de tipos de introducción de valores que ayuda a definir la interfaz del servicio.

2.4.9 Personalización

También la personalización, aspecto que influye en todas las fases de modelado, está cubierta de manera diversa en las diferentes aproximaciones. Algunas propuestas proporcionan marcos que complementan el modelo conceptual y dan soporte a rasgos de adaptación, adaptividad o incluso proactividad[11], como es el caso de W3I3 o UWE. Otras proporcionan únicamente mecanismos para la personalización de la aplicación en función de roles, como es el caso de OOHDM o WSDM, y por último hay algunas (como WAE) que ni siquiera la consideran.

[10] OO-H implementa esos mecanismos a nivel del propio metamodelo, tal y como veremos en el capítulo 7.

[11] La definición exacta de estos conceptos, junto con el modelo de personalización de OO-H, serán discutidos en el capítulo 7.

2.4.10 Soporte CAWE

Por último, prácticamente todas las metodologías y métodos estudiados hasta el momento reconocen la necesidad de un entorno de desarrollo para el éxito de implantación de la propuesta, y aportan herramientas que cubren en distinta medida los conceptos y proceso que plantean, tal y como hemos visto en la sección 2.3.

Sin embargo, el grado en que dichas propuestas son soportadas por su entorno varía ampliamente: mientras que algunas aproximaciones disponen de una CAWE completamente operativa, con capacidad de generar aplicaciones listas para su ejecución (como es el caso de OO-H y W3I3), otras (e.g. WAE) han extendido herramientas estándares para que soporten la notación, aunque no otros aspectos como es la consistencia entre modelos. Un trabajo similar de extensión de herramientas estándares pero que proporciona un soporte más completo del método está siendo realizado por aproximaciones como UWE, mientras que en otras (e.g. OOHDM, WSDM) su herramienta asociada actúa como mero editor de diagramas.

Soporte al Prototipado. Durante la fase de diseño, es interesante poder validar los caminos de navegación que van siendo diseñados, así como la estructura parcial de la interfaz. En este sentido, herramientas como WebRatio permite la generación parcial de código, mientras que otras (e.g. OO-H) anima los modelos dentro del propio entorno.

Soporte de Patrones y Guías de Diseño. Es un hecho reconocido que el uso de patrones genéricos (más allá de los cuatro patrones de navegación bien conocidos e incluidos como primitivas en la práctica totalidad de los métodos hipermediales existentes) mejora la calidad de las aplicaciones resultantes. Sin embargo, la especificación textual, poco sistematizada, de estos patrones, así como la complejidad de su almacenamiento y clasificación estandarizada, complican su utilización. Es por ello que herramientas como WebRatio (de W3I3) o la que acompaña al propio OO-H

proporcionan un modo de integración de dichos patrones en la herramienta para su aplicación automática sobre los modelos.

Generación automática código. Otro rasgo importante es el grado en el que cada entorno genera código final, y si es posible la generación para una o más plataformas. En general, todas las aproximaciones disponen de algún tipo de herramienta que almacena los modelos propuestos en su aproximación. Además, todas ellas abogan por la inclusión de compiladores de modelos que faciliten la tarea a los implementadores. Sin embargo, hasta lo que alcanza nuestro conocimiento, sólo W3I3 y OO-H proporcionan compiladores que generan aplicaciones operativas.

Soporte para propagación del cambio. Otro elemento importante en estos entornos CAWE es el soporte a cambios en los distintos modelos. Si estos modelos están relacionados (como es el caso normalmente), el entorno debería ser capaz de controlar esos cambios para cambiar las partes afectadas en consecuencia. Nuevamente hasta lo que alcanza nuestro conocimiento sólo W3I3 y OO-H, con herramientas propietarias, y UWE y WAE (por el soporte que proporcionan herramientas estándares como es el caso de Rational Rose) incluyen esta característica.

2.5 Uso de aproximaciones conceptuales en los entornos de trabajo actuales

No queremos acabar este capítulo sin alertar sobre los resultados de recientes estudios (Barry & Lang, 2001), que demuestran que el uso real de técnicas ingenieriles en el mundo empresarial para el desarrollo web y multimedial es francamente bajo. Resulta paradójico observar cómo este bajo uso contrasta con la actitud favorable que tienen los desarrolladores de este tipo de aplicaciones ante la adopción de algún tipo de metodología como modo de mejorar varios aspectos, entre los que destacan costes, velocidad de desarrollo, facilidad de comprensión o adaptabilidad, por poner algunos ejemplos.

Las principales razones argumentadas por las empresas consultadas son la falta de adecuación de las metodologías al mundo real y el largo período de formación necesario para su uso. Es por ello que actualmente las técnicas de los 70' son las más utilizadas. De hecho sorprende constatar el bajo porcentaje de empresas que utilizan modelos entidad relación y normalización de datos, que se reduce aún más si hablamos de aproximaciones más recientes orientadas al usuario y técnicas de modelado orientadas al objeto.

Desde este trabajo se apoya la tesis de que la estandarización de modelos y procesos y una orientación al usuario de los métodos y metodologías son elementos decisivos para facilitar su implantación en el entorno empresarial. Aún más importante, creemos que el desarrollo de herramientas CAWE que no sólo soporten el proceso de diseño sino que hagan hincapié en la gestión del proyecto (tarea crucial en el desarrollo comercial de productos) es un factor clave para potenciar la implantación de métodos hipermediales. Coincidimos con (Gellersen *et al.*, 1997) en que estos entornos deben cubrir el hueco existente entre el nivel de granularidad de los modelos de diseño (en continua evolución) y los modelos de implementación en web (idénticos desde su concepción), bien sea mediante el uso de técnicas de integración de componentes o mediante cualquier otro mecanismo, con el fin de asegurar la trazabilidad del cambio y mantenibilidad de la aplicación, y facilitar la ingeniería inversa. Este tema será retomado en el capítulo 9.

2.6 Conclusiones del Capítulo

En este capítulo se ha presentado una breve revisión de la problemática que genera el desarrollo de aplicaciones hipermediales mediante procesos poco estructurados, y cómo la comunidad hipermedial ha desarrollado diversas propuestas con el fin de dar respuesta a estos retos. Sobre ellas, se ha realizado un estudio comparativo atendiendo a aspectos generales (paradigma utilizado, uso de estándares, inclusión de un modelo de referencia, etc.) y específicos (relativos al proceso de autoría de la aplicación).

Desde este trabajo creemos que la paulatina resolución de las carencias detectadas dará lugar a la cuarta generación de aproximaciones hipermediales. Esta nueva generación se caracterizará, desde nuestro punto de vista, por proponer métodos y modelos altamente estandarizados, integrar un proceso preciso y disponer de un soporte CAWE completo. Además, todas las tendencias apuntan a que dichas aproximaciones incluirán modelos de análisis mucho más exhaustivos, donde probablemente el concepto de tarea tendrá un papel preponderante, incorporarán mecanismos de control de calidad e integrarán nuevas características de diseño que contribuyan a incrementar la velocidad de desarrollo al tiempo que aumentan dicho nivel de calidad, como es el caso de lenguajes de patrones. Todas estas características sin duda contribuirán a acelerar la transferencia de tecnología desde el ámbito académico en que nos movemos actualmente al mundo empresarial.

No obstante somos conscientes de que los métodos y procesos para el desarrollo de aplicaciones hipermediales tienen todavía un largo camino por recorrer. Estudios y trabajos comparativos como los realizados en este capítulo ayudan a identificar los puntos débiles y fuertes de los distintos métodos, y por tanto abren nuevas vías de investigación y facilitan la definición de procesos y métodos mejorados.

3. Fundamentos de OO-H: Características Generales y Proceso de Diseño

Todo método consiste en el orden y disposición de aquellas cosas hacia las cuales es preciso dirigir la agudeza de la mente.

DESCARTES

En el capítulo 1 introdujimos las razones que justifican la rápida implantación de las aplicaciones web en los entornos de trabajo actuales, los retos que este nuevo entorno plantea y los problemas que técnicas de desarrollo poco estructuradas están causando en la implantación y mantenimiento de este tipo de aplicaciones. En el capítulo 2 presentamos los avances y soluciones parciales que otras aproximaciones han dado a estos problemas en forma de metodologías, métodos, modelos y notaciones que abordan el desarrollo sistemático de aplicaciones hipermediales.

Este capítulo completa este primer bloque introductorio y presenta una visión general del método de modelado hipermedial conocido como *Método de modelado hipermedial orientado al objeto*[1] (OO-H), que fundamenta el presente trabajo. Comenzaremos justificando las características principales y el ámbito de aplicación de OO-H, para a continuación presentar, en el marco de un proceso de desarrollo adaptado a las características particulares de las interfaces hipermediales, el conjunto de modelos y técnicas que integran nuestra propuesta.

[1] Object-Oriented Hypermedia method

3.1 Objetivos de OO-H

OO-H (Gómez *et al.*, 2000; Cachero *et al.*, 2000d; Cachero *et al.*, 2000b; Gómez *et al.*, 2001a; Gómez *et al.*, 2001b; Gómez & Cachero, 2002) es un método genérico, basado en el paradigma objetual, que proporciona al diseñador la semántica y la notación necesaria para el modelado de interfaces basadas en web. Para ello OO-H define un conjunto de diagramas, técnicas y herramientas que se integran en un proceso de diseño flexible, en concordancia con la filosofía de *diseño del cambio* que presentábamos en el capítulo 1. El método captura modelos que posibilitan la generación automática de la interfaz final y su interconexión con módulos de lógica preexistentes.

OO-H resulta adecuado para su aplicación en el desarrollo de aplicaciones dinámicas, con una alta complejidad y estructuración de datos, donde la regularidad, la organización, la modularidad y la consistencia son factores clave, tales como intranets corporativas, aplicaciones de comercio electrónico, aplicaciones de educación a distancia, etc. Según algunos estudios (McClure, 1998) el 85% de las aplicaciones actuales adolecen de estas características, lo que justifica la importancia de desarrollar métodos y metodologías como las presentadas en este trabajo que sistematicen en lo posible su proceso de desarrollo.

Desde un punto de vista general, el objetivo de OO-H, lejos de ser el convertirse en *un método más*, se puede resumir en dos puntos:

- Integrar aquellos constructores, modelos y prácticas más relevantes propuestos en el marco de distintas aproximaciones (pertenecientes al campo hipermedial o campos afines) en una propuesta coherente, dentro de un proceso de desarrollo adaptado a la idiosincrasia de las interfaces web. Así, los modelos y el proceso de desarrollo de OO-H hacen uso de técnicas y conceptos tan generales como los Casos de Uso (UML 1.4, 2001), patrones hipermediales (Fraternali & Paolini, 1998), o constructores hipermediales ampliamente contrastados, como es el caso de colecciones o enlaces (Garzotto *et al.*, 1993).

- Aportar nuevas ideas y conceptos orientados a subsanar carencias detectadas en las propuestas hipermediales más relevantes de finales de los 90'. Entre estas nuevas ideas, presentadas en el capítulo 1 y que irán siendo desarrolladas a lo largo del presente trabajo, destacan la integración de funcionalidad compleja en los modelos de interfaz (en lugar de hablar sólo de consultas, altas, bajas y/o modificaciones), una propuesta de modelado de interfaces de servicio que incluye una taxonomía de tipos de parámetro o la descripción mediante un lenguaje estandarizado de las restricciones de navegación (Gómez *et al.*, 2000)[2].

OO-H resulta por tanto útil para:

- Modelado y generación completa de aplicaciones web cuya lógica se circunscribe al mantenimiento de los datos de la aplicación. En este caso OO-H es capaz de generar a partir del modelo conceptual de dominio los métodos correspondientes a este tipo de servicios de mantenimiento de datos. Es importante destacar que, debido a que OO-H no introduce modelos para especificar el comportamiento de dichos servicios, aspectos como su carácter transaccional o su activación mediante eventos queda fuera de las posibilidades de generación de OO-H.
- Modelado y generación completa de aplicaciones cuya lógica se encuentra encapsulada en módulos externos (procedimientos almacenados, dll's, javabeans, etc.) que deben ser integrados en una interfaz web de calidad. En este sentido, OO-H proporciona un conjunto de mecanismos para la especificación de los módulos que deben ser integrados en la aplicación, y su arquitectura permite generar el software intermedio para la interconexión con los distintos tipos de componentes externos, siempre que dispongamos del compilador de modelos adecuado.

Por otro lado es importante destacar que OO-H, al contrario que propuestas como WAE (Conallen, 1999) o UWE (Koch *et al.*,

[2] En este sentido, la creciente inclusión en otras aproximaciones de algunos de estos conceptos demuestran, desde nuestro punto de vista, lo acertado de las premisas que se encuentran en los orígenes de OO-H, y lo sitúan, tal y como vimos en el capítulo 2, en la vanguardia de los métodos existentes en la actualidad.

2001), no pretende ser una metodología que abarque el todo el ciclo de vida de las aplicaciones web. Así, aspectos como la planificación del proyecto (gestión de iteraciones, temporalización, etc.), la gestión de riesgos o la integración de procesos de evaluación de calidad no están contemplados como fases de OO-H. En contraste, OO-H se centra en las actividades de autoría de la aplicación: análisis de requisitos, ingeniería, construcción y adaptación y evaluación, tal y como veremos en la sección 3.4.

3.2 Características generales de OO-H

Dadas las características del entorno que presentamos en la sección 1.2.2, y con el fin de mejorar las capacidades del método como facilitador del desarrollo de interfaces hipermediales de calidad, OO-H se define como una aproximación:

- Dirigida por los requisitos de usuario
- Orientada a Objetos
- Parcialmente basada en estándares

A continuación veremos la motivación de cada una de estas decisiones.

3.2.1 El usuario como elemento central en OO-H

En el capítulo 1 destacamos cómo las aplicaciones web actuales compiten por captar la atención del usuario, y cómo la percepción que el usuario tiene de este tipo de aplicaciones influye de manera decisiva en su éxito de implantación. Es por ello que OO-H se separó desde sus orígenes de la tendencia clásica de aplicaciones dirigidas por los datos[3] y adoptó, siguiendo la tendencia de procesos de desarrollo de software más actuales, un enfoque orientado

[3] Estas aplicaciones también son conocidas como aplicaciones *data-driven*. En ellas el punto de partida es el esquema organizacional de los datos, y son las relaciones de dominio las que determinan la navegación a través de esos datos.

al usuario, donde la identificación y descripción de la audiencia (roles y requisitos de cada tipo de usuario que está previsto que acceda a la aplicación) dirige el resto de actividades de diseño. Esta aproximación contribuye a la producción de interfaces más ajustadas a necesidades individuales, y por tanto incrementa la facilidad de uso y el grado de satisfacción del usuario (Troyer, 2001)[4].

3.2.2 Aproximaciones Orientadas al Objeto: una concepción dinámica de la web

En el capítulo 2 comentamos cómo el modelo relacional está en los orígenes de muchas propuestas de modelado hipermedial. Este modelo resultaba perfectamente adecuado desde la perspectiva tradicional de *web como medio de diseminación de información*, ya que permitía expresar tanto la información de dominio como la funcionalidad necesaria para la consulta y el mantenimiento de esos datos (altas, bajas y modificaciones).

Sin embargo, la rápida evolución de las aplicaciones demandadas por las empresas ha puesto de manifiesto sus carencias (Gellersen & Gaedke, 1999; Manola, 1999; Ginige & Murugesan, 2001). Los modelos relacionales carecen de la potencia expresiva que ofrecen los modelos OO: atributos básicos del espacio del problema tales como encapsulación de datos y comportamiento, herencia, polimorfismo, identidad de objetos, referencias entre objetos (punteros), colecciones o tipos de datos definidos por el usuario no son expresables mediante esquemas relacionales, ya que éstos fueron pensados para modelar estructuras normalizadas físicas, bidimensionales, de datos, y en ellas se primó la rapidez de acceso frente a la potencia semántica (Robie & Bartels, 1994).

Otras ventajas de las aproximaciones orientadas a objetos son (Objectmatter, 2001):

[4] Esto no implica que OO-H no reconozca la valía de la estructura de los datos del dominio para incrementar la calidad de la interfaz; muy al contrario, esta información es utilizada en distintas fases del método para asegurar una coherencia semántica, tal y como veremos a lo largo del capítulo.

- Los objetos aíslan a los clientes de las reglas de negocio, del modo de almacenar datos físicamente y de factores relacionados con la concurrencia.
- Los objetos de negocio se basan en procedimientos previamente establecidos por la organización. Los expertos del dominio, que no son necesariamente desarrolladores, pueden diseñar el modelo.
- Los objetos de negocio pueden ser implementados como componentes reutilizables por varias aplicaciones. Los clientes pueden tener distintas vistas del mismo objeto dependiendo del propósito.
- Los modelos orientados a objeto favorecen la reutilización. De entre los mecanismos de reuso disponibles para este tipo de modelos destacan los patrones de diseño (Gamma *et al.*, 1995; Buschmann *et al.*, 1996).
- Las aplicaciones construidas con objetos de negocio adquieren una arquitectura abierta y son más fácilmente escalables. Los objetos se pueden implementar en una variedad de configuraciones, muchas veces siguiendo patrones de diseño ampliamente documentados (e.g. Observador[5] (Gamma *et al.*, 1995)). Este hecho las hace mucho más mantenibles, al separarse las distintas responsibilidades en capas con un bajo nivel de acoplamiento, lo cual además facilita su reuso y adaptación ante nuevos usuarios y/o nuevos dispositivos de acceso.
- Los modelos orientados a objetos incluyen mecanismos de seguridad mucho más evolucionados que los modelos relacionales.

Todos estos aspectos colocan a las aproximaciones OO en una posición de partida ventajosa para el modelado de aplicaciones hipermediales, al facilitar su adaptación a los cambios que se producen de manera continua en el entorno web.

[5] Observer

3.2.3 OO-H: un método basado en estándares

Por otro lado, y con el fin de facilitar la integración de OO-H dentro de procesos y herramientas comerciales que cubran aquellos aspectos que quedan fuera de su ámbito de aplicación, OO-H define modelos de entrada y salida estándares[6]. Además, el uso de estándares aporta otra serie de ventajas al método, como es la disminución de la curva de aprendizaje de desarrolladores, la mejora en la comunicación entre colaboradores de proyecto presentes y futuros, o la existencia de una buena documentación.

Los estándares utilizados en OO-H abarcan tanto lenguajes de especificación como protocolos de acceso[7], y entre ellos destacamos:

- Lenguaje Unificado de Modelado (UML) y el Lenguaje de Especificación de Restricciones (OCL) asociado.
- Lenguaje de marcado extensible (XML)[8] y estándares basados en él, como es el caso de Lenguaje de Descripción de Servicios (WSDL) [9] o el Lenguaje de Intercambio de Metadatos (XMI)[10].
- Protocolo de Acceso a Objetos Simple (SOAP)[11].
- Lenguaje de Patrones basados en la plantilla definida por Buschmann (Buschmann *et al.*, 1996) para la captura de los refinamientos más comunes de las interfaces hipermediales.

El motivo de haber seleccionado UML es su aceptación como estándar industrial *de facto* para modelar sistemas orientados a objeto. Además, UML aporta una serie de ventajas añadidas entre las que destacamos: (1) su extensibilidad mediante estereotipos, (2) la existencia de patrones de diseño, y (3) la integración de un lenguaje semiformal de especificación de restricciones (OCL) que

[6] Estos modelos y los correspondientes diagramas serán vistos en la sección 3.5
[7] El uso que OO-H hace de estos estándares será visto conforme se vayan presentado las distintas fases del método.
[8] eXtensible Markup Language (XML, 2000)
[9] Web Service Description Language (WSDL, 2001)
[10] (XML Metadata Interchange (XMI, 1999))
[11] Simple Object Access Protocol (SOAP, 2001) sobre HTTP

permite incrementar aún más su expresividad (Koch *et al.*, 2001), y que en OO-H es utilizado para la especificación de restricciones no sólo sobre el modelo conceptual sino también sobre el modelo de navegación.

Por otro lado, el carácter textual de XML y de las especificaciones basadas en él (como es el caso de XMI o WSDL) permite la representación de la semántica capturada en los modelos de OO-H mediante un lenguaje independiente de plataforma que además es fácil de validar contra una sintaxis predefinida, fácil de generar y fácil de leer. De especial relevancia para OO-H es XMI, que permite la exportación de dicha semántica a cualquier otra herramienta que soporte el estándar. Otras características de XML, relevantes desde el punto de vista de OO-H, son su extensibilidad y su falta de ambigüedad.

El uso de SOAP sobre HTTP como protocolo de invocación de servicios es otra de las ventajas de OO-H, ya que posibilita la interconexión con sistemas y componentes detrás de cortafuegos, siempre que dichos sistemas y/o componentes sean ofertados mediante una interfaz de Servicio Web.

Por último, OO-H proporciona un Catálogo de Patrones[12] en un formato estandarizado como mecanismo para encapsular conocimiento de diseño y facilitar el proceso de aprendizaje de diseñadores noveles. Este formato está basado en la propuesta de (Buschmann *et al.*, 1996), con la única adición de una sección que facilita la automatización de su aplicación sobre los modelos de OO-H.

Es importante destacar cómo estos rasgos no son invenciones de OO-H (ni de ninguna otra metodología hipermedial) sino que han sido fruto de un proceso de integración de prácticas ingenieriles preexistentes, tal y como comentamos a continuación.

[12] Este catálogo será presentado en el capítulo 8

3.3 La integración de perspectivas en OO-H

OO-H incluye técnicas, constructores y herramientas contrastadas provenientes de cada una de las distintas tendencias (hipermedial, bases de datos, modelos objetuales, componentes distribuidos) comentadas en la sección 1.4. Estamos de acuerdo con (Manola, 1999) en que cada una de estas tendencias resuelve de forma parcial los distintos matices de las aplicaciones hipermediales, y que por tanto se hace necesaria una fusión de sus respectivos puntos de vista para proporcionar una solución global y cohesiva al proceso de modelado de aplicaciones web.

Desde el punto de vista de la interfaz, los métodos de modelado hipermedial aportan una reflexión profunda acerca de aspectos de navegación e interacción usuario-sistema, rasgos básicos para que una aplicación hipermedial tenga éxito. Sin embargo una navegación y una presentación coherente no son suficientes, ya que las aplicaciones web deben también proporcionar al usuario la (a menudo compleja) funcionalidad que éste requiere, más allá de las consultas, altas, bajas y modificaciones que encontramos en el dominio de las bases de datos. Aún más, la estructuración de la aplicación debe ir dirigida a resolver los requisitos planteados por los distintos tipos de usuario, perspectiva que se opone a la orientación a datos de este tipo de aproximaciones.

En este sentido las aproximaciones tradicionales de Ingeniería del Software nos proporcionan mecanismos de probada eficacia para dotar a la aplicación de esta orientación al usuario, al mismo tiempo que nos permite modelar de manera conjunta estructura (datos) y comportamiento (funcionalidad), muchas veces en el marco de uno u otro tipo de entorno de producción automática de software avanzado, que integra técnicas de generación de código basadas en modelos (Bell, 1998; Gómez, 2000; Pastor *et al.*, 2001). El principal problema de este tipo de aproximaciones es que, utilizadas de manera aislada, carecen de mecanismos (modelos y constructores) que faciliten la captura de la semántica navegacional, que es precisamente el punto fuerte de los métodos de modelado hipermedial.

Por último, la funcionalidad de la aplicación puede estar centralizada en un solo servidor o distribuida en la red, y es aquí donde el concepto de *web de negocios dinámica* juega un papel relevante. Las propuestas actuales en este campo se plantean a un nivel de abstracción relativamente bajo. Estas propuestas de integración de servicios diseminados por la web en un marco aplicacional federado[13] se verían por tanto beneficiadas con un nivel de abstracción mayor, tal y como se plantea en aproximaciones de modelado tradicionales. También en sentido contrario, la inclusión de primitivas para la integración de servicios en el resto de los enfoques los capacitaría para hacer frente a la dinamización del entorno web de la que ya estamos siendo testigos.

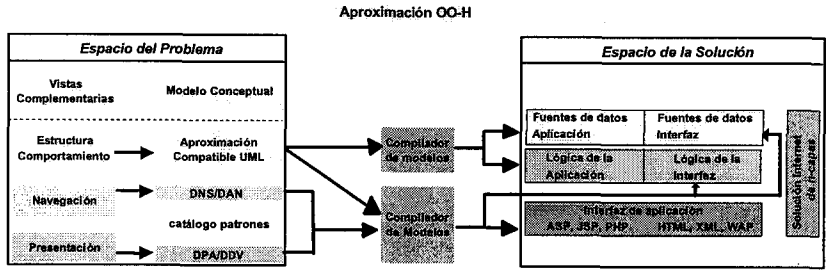

Figura 3.1. OO-H: una visión general

Tal y como podemos observar en la Fig. 3.1, para la materialización de esta integración, OO-H define un conjunto de vistas complementarias de la aplicación. El primer grupo lo constituyen las vistas que reflejan la estructura y comportamiento de la aplicación que, provenientes de la ingeniería del software tradicional, modelan en el espacio del problema tanto los requisitos funcionales del usuario como el dominio de la aplicación. Estas vistas constituyen el punto de entrada al método, y por ello se modelan en base a diagramas estándares UML, hecho que facilita la integración con otras herramientas y aproximaciones.

[13] Con este concepto nos referimos a sistemas que usan los servicios ofrecidos por otros sistemas, en un entorno donde el nivel de acoplamiento es sensiblemente menor que en el caso de los marcos de desarrollo basados en componentes (*Component-Based Development (CBD)*).

A partir de esta información, OO-H define, todavía en el espacio del problema, dos nuevos modelos particulares al ámbito hipermedial: navegación y presentación, para los cuales incluye un nuevo conjunto de constructores con semántica específica de este dominio hipermedial. La vista de navegación se materializa en un Diagrama de Navegación Semántica (DNS) y un Diagrama de Acceso Navegacional (DAN), mientras que la vista de presentación se materializa en un Diagrama de Presentación Abstracta (DPA)[14] y un diagrama de Diseño Visual (DDV)[15] (ver Fig. 3.1). El DCUA y el DAN capturan los caminos y las restricciones de navegación de cada tipo de usuario, incluido el modo que tiene de interactuar con los servicios ofertados[16]. Por otro lado el DPA y el DDV, tomando como base la información reflejada en el DAN, capturan a nivel lógico los conceptos relacionados con estructura y aspecto de cada página abstracta que compone la interfaz (nivel de presentación).

Por último, todavía en el espacio del problema, un Catálogo de Patrones agiliza la captura de información de diseño de la interfaz (tanto a nivel de DAN como de DPA y DDV), y constituye uno de los principales mecanismos de reuso que integra OO-H.

En la actualidad existen compiladores de modelos que, a partir de los modelos de estructura y comportamiento, son capaces de generar de manera automática las fuentes de datos y la lógica de la aplicación (Gómez, 2000; Pastor *et al.*, 2001). OO-H sigue la filosofía de estos entornos avanzados de producción automática de software, y proporciona un compilador de modelos que, partiendo de los modelos previamente diseñados, genera, ya en el espacio de la solución, la interfaz de la aplicación para la plataforma y en el lenguaje deseado (ver Fig. 3.1). Esta interfaz incluye: (1) la interfaz propiamente dicha, compuesta por un conjunto de páginas dinámicas y/o estáticas, (2) un módulo de lógica de interfaz y (3) un repositorio de información de interfaz. Estos dos últimos

[14] En realidad el DPA se encuentra en un punto intermedio entre la navegación y la presentación, tal y como veremos en el capítulo 8.

[15] Todos los diagramas que intervienen en OO-H serán descritos en la sección 3.5.

[16] En OO-H la invocación de un servicio también se considera navegación, como veremos en el capítulo 5.

componentes complementan los módulos de lógica y datos que pueden ser obtenidos a partir de las vistas tradicionales de estructura y comportamiento. Además, la generación de un módulo de lógica de interfaz y de un repositorio de lógica de interfaz permite la implementación características no presentes en la lógica de la aplicación, como pueden ser aspectos novedosos relacionados con la personalización o la seguridad del acceso. Todos estos módulos conforman una solución n-capa al modelado de aplicaciones web, y serán detallados en más profundidad en el capítulo 9.

En OO-H, la influencia del modelo de dominio en los modelos de navegación se materializa en clases y relaciones de dominio como fuente de información para la definición de dicha navegación. Es importante destacar que en OO-H estas relaciones sugieren (pero no condicionan, contrariamente a lo que ocurre en aproximaciones dirigidas por datos) caminos de navegación semánticamente relevantes. Por otro lado, el legado de aproximaciones basadas en componentes viene reflejado en el modelado explícito de interconexión con lógica preexistente.

Además, el modelado de las distintas vistas no se produce de manera anárquica, sino que sigue un proceso de desarrollo bien definido, que abordamos a continuación.

3.4 Proceso de desarrollo hipermedial en OO-H

Como ya comentamos en la sección 1.3.2, el desarrollo de software visible, repetible y medible requiere la sistematización de su proceso de desarrollo (R.S. Pressman, 2000). Es por ello que OO-H propone la integración de sus actividades en un marco de proceso en espiral presentado en la Fig. 1.1, que refleja el carácter eminentemente iterativo e incremental del método[17]. Este rasgo,

[17] Siguiendo la definición del Proceso Unificado de Rational Rose (RUP, 2001), un proceso iterativo es aquél que implica el manejo de un flujo de versiones ejecutables. Un proceso incremental por otro lado es aquél que implica la integración continua de la arquitectura del sistema para producir esas versiones, donde ca-

unido a la posibilidad de automatización de varios pasos del proceso, permite obtener productos parciales en etapas tempranas del desarrollo, lo que a su vez redunda en una mayor capacidad de adaptación al cambio.

Dentro de este proceso, OO-H define las actividades y artefactos que conforman el producto software, y se encarga por tanto de especificar aspectos relacionados con contenido, la navegación, la presentación y la funcionalidad que den respuesta a los requisitos funcionales capturados para dicha aplicación[18]. El objetivo de OO-H en este sentido es agilizar en lo posible la construcción de estos modelos, con el fin de poder producir prototipos funcionales en un tiempo relativamente corto, de acuerdo con la filosofía web de *diseñar para cambiar* que comentábamos en el capítulo 1.

Estas actividades, incluidas en el ciclo de vida de las aplicaciones que comentamos en la Fig. 1.1, y que conforman el proceso de autoría de aplicaciones en OO-H, son cuatro: (1) análisis de requisitos, (2) ingeniería, (3) construcción y adaptación y (4) evaluación del cliente. A continuación las describiremos brevemente, junto con el flujo de subtareas que abarca cada una de ellas.

3.4.1 Análisis de requisitos

El proceso de desarrollo en OO-H comienza con una etapa de especificación de requisitos funcionales similar a la que podemos encontrar en metodologías para el diseño de otros tipos de aplicaciones software. Esta etapa de recogida de requisitos para cada tipo de usuario dirige el resto de fases del proceso de desarrollo, tal y como veremos en el capítulo 4.

---

da nueva versión incluye mejoras incrementales sobre la anterior (Booch *et al.*, 1999).

[18] Esto implica que aspectos como la captura de requisitos no funcionales, la gestión del riesgo, el número de iteraciones del proceso o la gestión del mantenimiento/evolución de la aplicación, por nombrar algunos, no son tratados en OO-H.

3.4.2 Ingeniería

Dentro de la etapa de ingeniería se engloban todas las actividades relacionadas con el análisis y diseño del producto software, que son:

Análisis de dominio. OO-H, al igual que el resto de aproximaciones hipermediales, incluye una actividad de análisis conceptual, donde, a partir de los requisitos de usuario y del conocimiento que el diseñador tenga del dominio, se capturan los conceptos relevantes para la aplicación.

Análisis de navegación. Paralelamente al análisis de dominio, OO-H incluye una actividad de análisis de navegación. Esta actividad proporciona una estructuración de la interfaz basada en un análisis de las relaciones entre las distintas tareas de usuario, que no tienen por qué seguir los mismos criterios estructurales aplicados durante el análisis de dominio. De este modo es el conocimiento de los requisitos de usuario y no el conocimiento de estructuración de conceptos a nivel de dominio el que dirige las decisiones de alto nivel acerca de la estructura de navegación de cada tipo de usuario, lo cual a su vez mejora la facilidad de uso del sistema.

Diseño de dominio. Según se va avanzando en la especificación, el modelo de análisis de dominio debe ir siendo refinado en iteraciones sucesivas con nuevas clases de apoyo[19], tipos de atributo, parámetros de los métodos, etc. Además, se debe tener especial cuidado en comprobar la exactitud de las interfaces de componentes de lógica/bases de datos externas (si es que existen), ya que esta información será básica para la correcta generación de los módulos de conexión en la etapa de construcción de la aplicación.

Diseño de navegación. El análisis de la navegación y la información de dominio constituyen las entradas principales para la actividad de diseño de navegación, donde se determinan los caminos que pueden ser recorridos a través del espacio de información

[19] *helper classes*

para cubrir cada uno de los requisitos funcionales, así como la agrupación de dicha información en páginas abstractas[20]. Es importante destacar que, como justificaremos en el capítulo 5, la llamada a servicios de lógica es considerada navegación en OO-H, y por tanto la captura de la interacción del usuario con los distintos servicios ofertados forma también parte de esta actividad, en la que se determina el modo en que éste va a introducir los parámetros de entrada y cómo va a visualizar los resultados devueltos por la lógica (parámetros de tipo *in* y *out* respectivamente).

Diseño de Presentación. Una vez capturada la estructura lógica de la interfaz mediante la actividad de diseño de la navegación, OO-H permite especificar localización, apariencia y componentes gráficos adicionales que van a materializar la información y la navegación contenida en cada una de las páginas abstractas. Seguimos hablando de diseño porque los constructores de esta etapa son componentes genéricos (independientes de plataforma), y que deberán por tanto sufrir un proceso de transformación adicional en la etapa de implementación para su funcionamiento sobre la plataforma y tecnología consideradas.

El análisis y diseño de dominio serán tratados con más profundidad en el capítulo 4. El análisis y diseño de navegación serán desarrollados en el capítulo 5. Por último, el diseño de la presentación será abordado en el capítulo 8.

3.4.3 Construcción y adaptación

Una vez dada por finalizada la fase de ingeniería, OO-H solicita una serie de datos relacionados con la implementación, que tendrán que ser considerados a la hora de generar la interfaz web. Estos datos se complementan con un conjunto de plantillas que contienen conversiones estándar aplicables a los modelos para permitir la generación de código final. A continuación damos una

[20] Una página abstracta es una unidad lógica de agrupación de contenido y enlaces, y por tanto no tiene por qué corresponder con una página física tal y como la ve el usuario final en el navegador. Este concepto será ampliado en el capítulo 8.

visión general de las principales actividades que conforman esta fase.

Arquitectura de ejecución. El primer grupo de parámetros necesario es el que se refiere al tipo de arquitectura que se pretende generar en función de la plataforma de implementación final, y que puede variar en cuanto a número de capas. Dentro de esta fase, destaca la asignación de distintas responsabilidades al cliente y al servidor.

Integración. El segundo grupo de parámetros importantes es el referido a ubicación física, protocolos de acceso, etc. de los módulos de lógica con los que debe interactuar la interfaz. En el estadio actual de desarrollo de OO-H, esta integración presupone que todas las fuentes de datos y servicios externos se ofrecen como Sevicios Web, con la restricción adicional de estar ubicados en un lugar predefinido[21]. Esta información es la que debe ser utilizada para generar el módulo mediador presentado en la Fig. 3.1.

Configuración de parámetros de generación. El compilador de modelos también necesita saber el tipo de tecnología que se desea utilizar (lenguajes cliente y servidor, utilización o no de marcos, envío o no de *cookies* al cliente, etc.) para, en función de ellos, realizar una conversión estándard (basada en plantillas preexistentes en el entorno OO-H) que, partiendo de los constructores de interfaz abstractos que incluye OO-H como parte de su modelo de presentación[22], asocia los constructores físicos correspondientes al lenguaje de implementación deseado. Por ejemplo, estas plantillas son las encargadas de determinar el tipo de constructor físico (e.g. una etiqueta <SELECT> en un entorno HTML) que debe ser generado si, a nivel de diseño de presentación, nos encontramos con un constructor abstracto de tipo *lista de opciones.*

Generación. Por último, con toda la información recogida, el compilador está en disposición de generar la aplicación final.

[21] El soporte completo de los Servicios Web, incluyendo mecanismos de búsqueda de servicios, queda como trabajo futuro.

[22] *widgets*

El modo en que el entorno de desarrollo de OO-H implementa estos pasos y permite la generación e implantación del sistema será visto en el capítulo 9.

3.4.4 Evaluación

OO-H, al contrario que otros métodos hipermediales, incluye dos tipos de actividades de evaluación: una actividad de prototipado, que puede ser efectuada dentro del entorno de desarrollo de OO-H sin necesidad de haber completado los modelos, y una actividad de validación que se debe realizar tras la generación de cada versión de la aplicación.

Prototipado. OO-H permite simular la estructura y comportamiento de la interfaz modelada sin tener que generar una sola línea de código. La ventaja de este prototipo animado es que, debido a que se ejecuta dentro del entorno de desarrollo de OO-H, no es necesario que los modelos estén completos, lo que permite validar cada decisión de diseño y realizar los cambios necesarios en etapas tempranas de desarrollo, con la correspondiente reducción de costes. Es importante destacar que este prototipo no es funcional, es decir, no incluye datos reales ni conexiones con bases de datos o componentes externos, sino que se limita a mostrar la estructura (los caminos de navegación) y la apariencia de la interfaz diseñada hasta el momento. A pesar de que esta tarea es claramente una actividad de evaluación, OO-H propone que se realice durante la fase de diseño. Experiencias empíricas demuestran que de esta manera (posibilitando que el diseñador visualice algún tipo de resultado antes de llegar a la fase de generación) se reduce el número de iteraciones necesarias para modelar la interfaz final, y por tanto se aumenta la eficacia del método.

Validación. Por otro lado, una vez generado el código de la interfaz deseada, una actividad de validación por parte del diseñador permite comprobar la corrección de aquellos aspectos que no pueden ser comprobados mediante el entorno de prototipado: funcionalidad, conexión con datos y servicios externos, etc. El objetivo

en este paso es asegurar que la aplicación final (el producto que se le va a entregar al usuario) cumple con todos los requisitos especificados en las primeras fases del método, tal y como los ha entendido el diseñador. En este punto es importante destacar cómo el concepto de validación por parte del diseñador incluye tanto la comprobación manual de la corrección de sus especificaciones (e.g. asegurarse de que todos los requisitos de usuario previstos pueden ser cubiertos mediante los caminos de navegación incluidos en el modelo de navegación diseñado) como la comprobación automática (ayudado por herramientas integradas en el entorno de desarrollo de la aplicación) de la corrección del código generado, e.g. pruebas de alcanzabilidad de los enlaces externos definidos en el modelo de navegación. Por último, OO-H contempla la posibilidad de definir una serie de criterios de calidad que también deberán ser validados en esta fase. Este aspecto queda como trabajo futuro, y por tanto fuera del alcance de la presente tesis.

Por último, en relación a esta actividad nos gustaría destacar cómo el hecho de disponer de un compilador de modelos que genera de manera automática la aplicación a partir de los modelos OO-H[23] simplifica el trabajo de validación, ya que evita gran cantidad de errores de implementación, y permite asegurar la compatibilidad de la interfaz para los entornos de ejecución deseados.

La última tarea dentro de esta etapa de evaluación es la validación por parte del cliente, es decir, la constatación de que la aplicación es y hace exactamente lo que él esperaba. El modo en que se efectúe esta tarea queda fuera del control de OO-H. Sin embargo, sea cual sea la técnica empleada, el resultado obtenido en esta actividad sí debe ser introducido en OO-H en forma de refinamientos y/o nuevos requisitos que serán los que guíen la siguiente iteración del método. Nuevamente, el entorno de prototipado permite dividir esta tarea en dos pasos: un primer paso antes de generar la aplicación, y que se centra en la validación de contenido, navegación y presentación de la interfaz, y un segundo

[23] Este compilador será tratado en profundidad en el capítulo 9.

paso, una vez generada la aplicación, que se centra principalmente en la validación de la funcionalidad.

3.4.5 Personalización, Seguridad, Calidad y Documentación

Todas las actividades mencionadas anteriormente no se realizan de forma aislada, sino que en todas ellas juegan un papel más o menos preponderante las necesidades de personalización, seguridad, calidad de la solución ofertada y documentación de las decisiones de diseño adoptadas.

La identificación temprana de las necesidades de personalización disminuye el coste de modificación de las aplicaciones, y supone un factor importante para el éxito de implantación de las aplicaciones hipermediales, como demuestran lugares web en explotación como Amazon o las oficinas de banca electrónica. Estos requisitos de personalización incluyen cambios en el contenido, la navegación o la apariencia de la interfaz causadas por variables como el tipo de usuario, su actividad previa, su localización o cualquier otro aspecto que tenga que ver con sus condiciones particulares. Este concepto, así como su influencia en los distintos modelos de OO-H, será tratado en profundidad en el capítulo 7.

Otro aspecto importante en el ámbito de las aplicaciones web es, tal y como vimos en el capítulo 1, la seguridad de acceso. Este tema, debido a su complejidad, queda fuera del alcance de este trabajo, pero creemos importante destacar que su consideración puede afectar a la propia naturaleza de la aplicación, y por tanto, al igual que ocurría con la personalización, una consideración temprana de las necesidades y riesgos involucrados puede disminuir de forma drástica los costes en los que se incurriría en caso de posponer su consideración.

Por otro lado existe un amplio campo de trabajo relacionado con el uso de patrones[24], marcos de trabajo, consejos de diseño[25], etc.

[24] Los patrones son soluciones recurrentes y efectivas a problemas de diseño bien documentados.

[25] *golden rules* (Nanard *et al.*, 1998)

para incrementar la calidad de los modelos y, consecuentemente, de la propia aplicación diseñada. OO-H reconoce su utilidad y por tanto proporciona un Catálogo de Patrones que captura estos conceptos. Las principales características de este catálogo son, por un lado, su adhesión a estándares bien documentados, y por otro su extensibilidad, lo que permite que dicho catálogo evolucione según se van identificando nuevas soluciones reutilizables. Además, y debido a que la especificación de patrones de OO-H ha sido extendida para incluir una sección que implementa los cambios que su aplicación introduce en los distintos modelos, este catálogo no sólo asegura la calidad de la solución aportada, sino que además agiliza el desarrollo de la aplicación, constituyéndose en un verdadero lenguaje de diseño. Una visión exhaustiva del Catálogo de Patrones existente en OO-H será presentada en el capítulo 8.

Por último, es importante destacar la importancia de una buena documentación de las decisiones de diseño adoptadas en cada fase del proceso. En este sentido, nuevamente el uso de patrones documenta implícitamente muchas decisiones de diseño de forma exhaustiva. Además, OO-H incluye como característica del entorno de generación que da soporte al método la capacidad de anotación, que posibilita la captura en lenguaje natural de información adicional acerca de las decisiones de diseño adoptadas. Por último, cada una de las principales actividades de OO-H genera o refina un modelo, plasmado en uno o varios diagramas que, entre otras atribuciones, se encargan de documentar las decisiones adoptadas, tal y como veremos a continuación.

3.5 Modelos y diagramas de OO-H

En la sección 1.3.1 definimos el concepto de modelo como *un producto final, generado como resultado de la ejecución de una o varias actividades.* También dijimos que *los modelos definen una serie de normas en la forma de uso del lenguaje de modelado, y por tanto en la forma de ejecutar la actividad.* Cada modelo en

OO-H refleja por tanto una vista de la interfaz que pretendemos generar, y surge como resultado de ejecutar una o más de las actividades que presentábamos en la sección 3.4.

En general, los modelos son un poderoso mecanismo de comunicación. En OO-H no sólo sirven para documentar el dominio del problema o las decisiones de diseño, sino que la trazabilidad[26] existente entre los mismos permite predecir el impacto de los cambios realizados en cualquier nivel (contenido, navegación, presentación, funcionalidad) sobre el resto de la interfaz.

OO-H propone los siguientes modelos y sus correspondientes entregables (diagramas):

- Modelo de Requisitos Funcionales (Diagrama de Casos de Uso, compatible UML), obtenido como resultado de la actividad de análisis de requisitos
- Modelo de Dominio (Diagrama de Clases, compatible UML), obtenido como resultado de las actividades de análisis y diseño de dominio
- Modelo de Análisis de Navegación (Diagrama de Navegación Semántica), obtenido como resultado de la actividad de análisis de navegación
- Modelo de Diseño de Navegación (Diagrama de Acceso Navegacional), obtenido como resultado de la actividad de diseño de navegación
- Modelo de Diseño de Presentación (Diagrama de Presentación Abstracta, Diagrama de Diseño Visual), obtenido como resultado de la actividad de diseño de presentación

Estos modelos sirven de entrada a la fase de construcción y adaptación, donde se produce la generación de la aplicación modelada.

A continuación veremos con detalle cada uno de ellos. Con el fin de ilustrar todo el proceso, una representación gráfica de la

[26] Definimos trazabilidad como la habilidad de comenzar con un elemento de un modelo y trazar sus colaboraciones y conexiones con otras partes del mismo u otro modelo (Conallen, 1999)

secuencia lógica de diagramas dentro del proceso de desarrollo de OO-H puede ser observada en la Fig. 3.2.

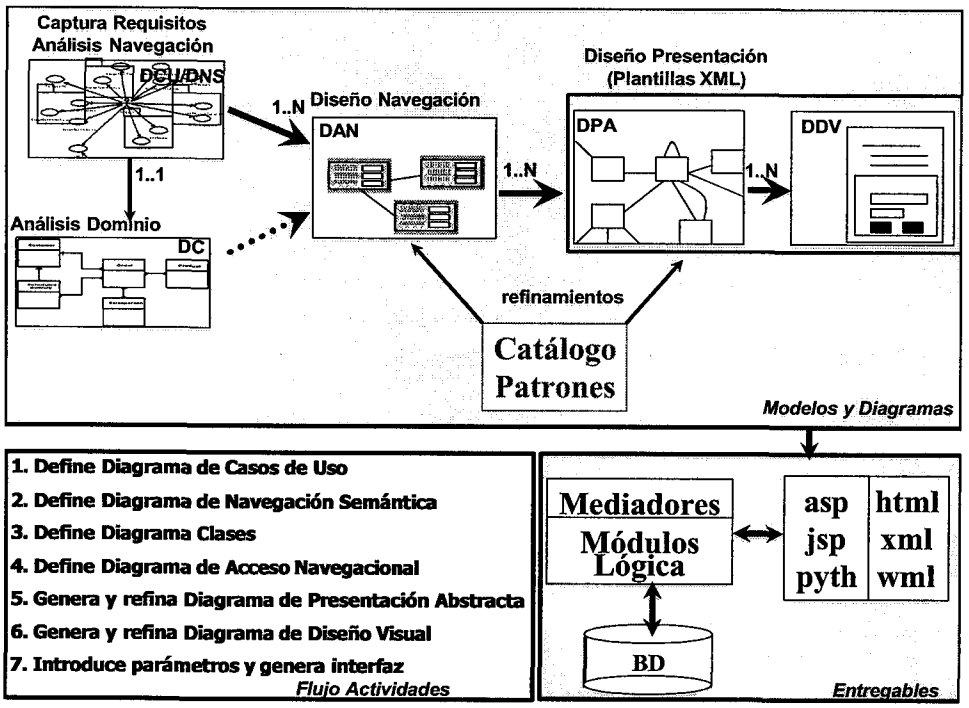

Figura 3.2. Modelos de aplicación en OO-H

3.5.1 Modelo Requisitos Funcionales

En la Fig. 3.2 podemos ver cómo la primera actividad del proceso de desarrollo de la interfaz hipermedial, tal y como presentábamos en la sección 3.4, es la captura de requisitos de usuario. Esta actividad, cuya técnica de ejecución entra dentro del campo de la Ingeniería de Requisitos y por tanto fuera del ámbito de este trabajo, proporciona como resultado un diagrama de casos de uso compatible UML, que constituye el modelo de requisitos funciona-

les. Este diagrama se adapta perfectamente al enfoque orientado al usuario que guía toda nuestra aproximación.

3.5.2 Modelo Análisis Navegación

Sobre el diagrama de Casos de Uso resultante de la actividad de análisis de requisitos, el Análisis de Navegación introduce un conjunto de enriquecimientos que dan lugar a un modelo materializado en un Diagrama de Navegación Semántica (DNS), tal y como podemos ver en la Fig. 3.2. El modelo de análisis de navegación captura por tanto la estructuración de más alto nivel de la navegación del usuario en el sistema, y su existencia obliga al diseñador a tener en cuenta el posible propósito de dicho usuario a la hora de diseñar los menús de acceso a los distintos requisitos, evitando de este modo que la interfaz acabe siendo dirigida por la estructura de la información de dominio, tal y como ocurre en otras aproximaciones. La necesidad de un modelo de análisis de navegación y sus características será visto en profundidad en el capítulo 5.

3.5.3 Modelo Dominio

El modelo de dominio de OO-H se materializa en un diagrama de clases (DC) compatible UML, que incluye clases, atributos, métodos y relaciones del dominio del problema. Este modelo surge como resultado de varias actividades. En primer lugar, un análisis de dominio captura los objetos y relaciones que colaboran en dar respuesta a los requisitos de la aplicación. Según vamos avanzando hacia el diseño, este DC es refinado para recoger detalles de más bajo nivel, como es el tipo de los atributos, parámetros de entrada y salida de los métodos, precondiciones y postcondiciones de invocación, etc. Otros refinamientos que tienen lugar durante la fase de diseño, tal y como veremos en detalle en el capítulo 4, son la aplicación de patrones de diseño y la captura de interfaces con módulos de lógica preexistentes.

3.5.4 Modelo Diseño Navegación

Toda la información contenida en los modelos anteriores sirven de base para la construcción de un modelo de diseño de navegación, que representa el contenido, la navegación y la funcionalidad que componen la interfaz de la aplicación. El modelo de diseño de navegación de OO-H, eje central de este trabajo, tiene como objetivo principal la captura, mediante constructores hipermediales, de toda la semántica (contenido y navegación) necesaria para permitir la generación de un primer prototipo que permite validar la navegación del usuario a través de la interfaz. El diagrama de acceso navegacional (DAN) asociado recoge la información sobre los caminos navegacionales, restricciones de navegación e interfaces de servicio, que tienen como fin el dar respuesta a cada requisito funcional del sistema.

Este modelo y su correspondiente diagrama serán discutidos en profundidad en el capítulo 5.

3.5.5 Modelo Diseño Presentación

El último modelo incluido en OO-H es el modelo de Diseño de Presentación, correspondiente a la actividad homónima. Este modelo incluye dos diagramas: por un lado el Diagrama de Presentación Abstracta (DPA) presenta una visión de la estructura lógica de la aplicación, y permite modificar dicha estructura con elementos como cabeceras y pies, páginas estáticas etc. Por otro lado el Diagrama de Diseño Visual (DDV) permite editar cada una de las páginas que conforman la estructura para especificar aspectos como colores, posicionamiento o tipos de constructor lógico. Es importante resaltar que esta especificación no es dependiente de dispositivo ni de plataforma final, por lo que los rasgos capturados requieren un proceso de traducción durante la fase de generación.

3.6 La herramienta CAWE y el compilador de modelos

El conjunto de perspectivas aportadas por los distintos modelos conforman la descripción de la interfaz, y son traducidos a una especificación XML que es utilizada como entrada del compilador de modelos. Este rasgo es fundamental, ya que permite la reutilización del módulo compilador fuera del ámbito del entorno de desarrollo de OO-H (ver Fig. 3.1), siempre que se preserven la estructura de documento recogida en los distintos DTD's.

Además, la existencia de este compilador de modelos aporta las siguientes ventajas:

- Acelera el proceso de desarrollo
- Elimina actividades susceptibles de generar errores
- Se facilita el reuso y la migración a nuevos entornos/dispositivos de acceso (sólo requiere implementar un nuevo traductor en la CAWE, y evita los tediosos testeos de compatibilidad).
- Permite desviar el foco de atención de los desarrolladores hacia las actividades de análisis y diseño, que son la fuente de muchos de los problemas asociados con el proceso de desarrollo (Jayaratna, 1990).
- Se asegura una interfaz con una filosofía homogénea: tipos de comprobaciones, tipo y formato de mensajes de error, comportamiento ante la invocación de servicios, mecanismos de seguridad, etc. se mantienen a lo largo de toda la aplicación, e incluso entre plataformas.
- Incrementa la calidad de ajuste: la concordancia de la interfaz final tanto con respecto a los modelos de las fases de análisis y diseño como con respecto a la documentación del sistema queda garantizada.

A pesar de que este compilador de modelos puede ser utilizado independientemente del entorno que da soporte a los modelos

OO-H, el desarrollo de aplicaciones se optimiza si se utilizan de manera conjunta con dicho entorno de desarrollo. En efecto, la herramienta CAWE que acompaña a OO-H proporciona no sólo la capacidad de generación automática de código, sino que también aporta capacidades de cambio automático de vistas, soporte para la aplicación de patrones de diseño y automatización de parte de las actividades de diseño. Todo ello es necesario si se aspira, como es el caso de OO-H, a facilitar la implantación del modelado conceptual de aplicaciones web, que en la actualidad sigue siendo minoritario (Barry & Lang, 2001).

Por último, ya hemos comentado cómo el compilador, en la fase de construcción y adaptación, solicita la información necesaria para la generación automática de la interfaz de la aplicación. Antes de proceder a dicha generación, la herramienta CAWE realiza un nuevo tipo de testeo sobre las páginas generadas: servicios accesibles, links externos alcanzables, etc. Si todo es correcto, el compilador genera una serie de módulos de código, cuya arquitectura pasamos a describir a continuación.

Arquitectura de las aplicaciones OO-H. En OO-H el *modelo de arquitectura web* se define como la descripción de los componentes básicos de su implementación, y cómo se conectan para soportar la funcionalidad e interacciones requieridas con el usuario (Paterno & Mancini, 1999). Esta definición se separa así de la perspectiva tradicional, donde con él se hacía referencia a la estructura navegacional (linear, jerárquica, en rejilla, etc.) de la interfaz (R.S. Pressman, 2000).

Los componentes (entidades autónomas con una interfaz propia) relevantes de la aplicación son, como mínimo (Conallen, 1999): (1) un navegador en uno o más clientes que se comunican con un servidor mediante el protocolo de transferencia hipertextual[27] y (2) un servidor de aplicaciones que se encarga de gestionar la lógica de la aplicación. Sin embargo, desde este trabajo abogamos por un enriquecimiento de este conjunto mínimo de componentes para conseguir, de acuerdo con la filosofía de *preparar para el cambio*

[27] HyperText Transfer Protocol (HTTP)

que subyace tras el concepto de Ingeniería Web, una arquitectura (lo que algunos autores llaman *modelo de producto* (Ginige, 1998)) lo más flexible posible. Así, el modelo de arquitectura en OO-H tiene la forma que se presenta en la Fig. 3.3.

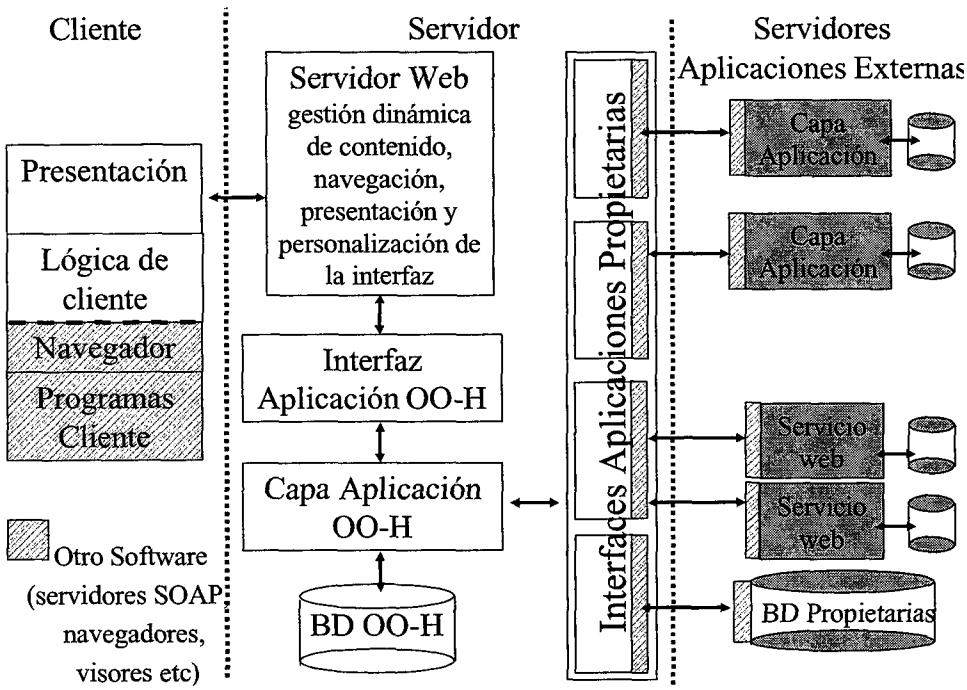

Figura 3.3. Arquitectura de las aplicaciones generadas mediante OO-H

En este modelo podemos observar una parte cliente y una parte servidora. En la parte cliente existe una separación entre los aspectos de *presentación* visual y aspectos de *lógica de cliente* relacionados con los constructores de presentación (e.g. reacción ante eventos de usuario, control de tipos de entrada, etc.). Este cliente se comunica con el/los servidores web a través de la red. Los servidores albergan las páginas que constituyen la *interfaz de la aplicación*. Estas páginas materializan la navegación a través de las estructuras de información, y por tanto disponen del conocimiento necesario para comunicarse, a través de un módulo de

lógica intermedia (*capa de aplicación OO-H*) tanto con la *base de datos OO-H*, generada desde el propio entorno de desarrollo, como con *módulos de lógica, bases de datos o servicios web externos* (de los que OO-H sólo conoce su interfaz, sus protocolos y parámetros de conexión). Las bases de datos generadas a partir de los modelos OO-H complementan a las (posibles) bases de datos externas y permiten a OO-H enriquecer la aplicación con aspectos tales como capacidad de personalización o reglas de negocio.

El catálogo de patrones en la CAWE. Con respecto a la aplicación de patrones dentro del entorno de desarrollo de OO-H, todos los patrones incluidos en el Catálogo contienen como parte de su especificación de implementación una sección adicional denominada *Regla de Transformación*[28], un fragmento de código que permite la aplicación de dichos patrones sobre los distintos modelos y constructores de OO-H (contenidos en el modelo de referencia del método). Este catálogo será visto con detalle en el capítulo 8.

La documentación en la CAWE. Por último, ya hemos comentado cómo OO-H proporciona distintos mecanismos de documentación, que son:

- Los propios modelos.
- Registro de los patrones que han sido aplicados en cada modelo.
- Mecanismos de anotación que permiten la especificación, en lenguaje natural, de restricciones no consideradas explícitamente en los modelos.

A todos estos mecanismos el entorno de desarrollo de OO-H añade un generador de documentación que, a partir de los modelos, permite la inclusión de toda la información recopilada dentro de la propia interfaz generada. Esta información (incluidos los diagramas relativos a la estructuración de la interfaz) constituye otro mecanismo para incrementar la facilidad de uso de la aplicación al mismo tiempo que simplifica su posterior mantenimiento.

[28] *Transformation Rule*

3.7 Conclusiones del capítulo

OO-H es un método de modelado de aplicaciones hipermediales, caracterizado por su orientación al objeto, el uso de estándares y un proceso de diseño dirigido por requisitos de usuario. Este proceso de diseño define además una separación clara de los distintos modelos, entre los que existe una trazabilidad bien definida que no sólo facilita la evolución de la aplicación web sino que además es una base sólida para maximizar el reuso de elementos de diseño e implementación.

En este sentido OO-H define toda una serie de mecanismos que promueven el reuso y, por tanto, incrementan la velocidad de desarrollo de las aplicaciones, característica fundamental en un entorno tan dinámico como la web, y que son:

- Trazabilidad entre los distintos modelos del método, que separan de manera taxativa los distintos niveles de abstracción (datos-funcionalidad-navegación-presentación). Esta trazabilidad posibilita la existencia de una automatización parcial de las actividades de modelado que incrementa la velocidad de desarrollo, fundamental en este tipo de aplicaciones de ciclo de vida normalmente mucho más corto.
- Características de prototipación desde fases tempranas del método
- Catálogo Patrones
- Herramienta CAWE y Compilador de Modelos

OO-H plantea un proceso de desarrollo que, partiendo de los requisitos funcionales del usuario, propone un conjunto de actividades y técnicas que generan como resultado una descripción exhaustiva de la interfaz web que se desea construir. Este proceso de desarrollo se descompone, al igual que el proceso de desarrollo de aplicaciones tradicionales, en cuatro fases iterativas e incrementales, que son:

- Análisis de Requisitos, donde el diseñador establece las bases del método. Este modelo es común a muchas aproximaciones, hipermediales o no.
- Ingeniería, donde se definen qué objetos de dominio son relevantes para cubrir esos requisitos y cómo se debe desarrollar a grandes rasgos la aplicación. Los modelos generados como resultado de este análisis son altamente reutilizables, al ser independientes de plataforma y/o dispositivo de acceso. Además, los modelos de análisis de requisitos y análisis y diseño de dominio están basados en estándares (UML, OCL), lo cual facilita su comprensión e intercambio entre entornos CAWE[29].
- Construcción y Adaptación, donde se introduce la información relativa al entorno de ejecución. Esta información posibilita la generación, mediante un compilador de modelos, de la interfaz de la aplicación y sus interconexiones con datos y lógica subyacente. Estos parámetros incluyen arquitectura de ejecución deseada, lenguaje y plataforma objetivo, etc.
- Evaluación, donde se valida la corrección y calidad de la interfaz generada. El resultado de esta actividad es un conjunto de requisitos y refinamientos que constituyen la entrada de la siguiente iteración.

Además, este capítulo ha identificado los aspectos que no son abordados en OO-H, y para los que por tanto se requiere el uso de técnicas propuestas en otros ámbitos, como es el tratamiento de los requisitos no funcionales (seguridad, privacidad, rendimiento, etc.), planificación de iteraciones o análisis de riesgos. Para todos estos aspectos, OO-H proporciona de momento simplemente mecanismos de anotación en los modelos.

En resumen, las contribuciones más relevantes de este capítulo son las siguientes:

[29] En realidad OCL es utilizado también en el modelo de navegación de OO-H, aunque de una manera no estandarizada (i.e. asociado a enlaces, y no a clases u operaciones). Este uso será ilustrado en el capítulo 5.

- Enumeración de los aspectos más significativos de nuestra propuesta.
- Definición del proceso de desarrollo de interfaces hipermediales en OO-H.
- Presentación de los distintos modelos que fundamentan nuestra propuesta.
- Arquitectura de las interfaces de aplicación resultantes.

Volver al Índice/Tornar a l'índex

4. Análisis de Requisitos y Modelado de Dominio en OO-H

Si no sabes a dónde vas, acabarás en otra parte.

L. PETER

Como ya comentamos en el capítulo 3, OO-H plantea un proceso de desarrollo que, partiendo de los requisitos funcionales del usuario, propone un conjunto de actividades y técnicas que generan como resultado una descripción exhaustiva de la aplicación web que se desea construir. Este proceso de desarrollo se descompone, tal y como vimos en el capítulo 3, en las siguientes actividades: (1) Recogida de requisitos, (2) Ingeniería (Análisis de navegación, Análisis de dominio, Diseño de Dominio, Diseño de navegación y Diseño de presentación), (3) Construcción y Adaptación (que culmina con la generación del código de la aplicación) y (4) Evaluación (prototipado y validación de la aplicación).

En este capítulo nos centraremos en definir las actividades necesarias para comprender tanto los objetivos (requisitos funcionales) que debe cumplir la aplicación como el dominio (espacio de información) en que se desenvuelve. Este conocimiento es la base sobre la que se asientan las actividades relacionadas con la navegación del usuario a través de dicho espacio de información, que constituyen la principal aportación del presente trabajo y que serán presentadas en los capítulos 5, 6 y 8.

Antes, presentaremos la descripción de un *Sistema de Gestión de Hoteles (SGH)*, que utilizaremos a partir de ahora para ejemplificar y clarificar los distintos conceptos involucrados en nuestra aproximación.

4.1 Caso de Estudio: Sistema de Gestión de Hoteles

El objetivo de esta aplicación es gestionar las reservas y facturación de habitaciones de un hotel. Las habitaciones pueden ser de distintos tipos: suites, habitaciones dobles con cama de matrimonio, habitaciones dobles con camas gemelas, habitaciones individuales, etc. Es precisamente el tipo de habitación el que determina su precio de reserva. El sistema ofrece a sus clientes registrados la posibilidad de realizar una reserva sobre la habitación que elijan. Cada reserva viene definida por su fecha de llegada y de partida, con la restricción de que, sobre una misma habitación, no pueden existir dos reservas que se solapen.

El hotel ofrece una serie de servicios (lavandería, distintos productos de minibar, peluquería, etc.) con una tarifa. Estos servicios se registran como cargos asociados a la habitación cada vez que el cliente hace uso de ellos. A la hora de facturar, el sistema registra el método de pago utilizado, y permite la inclusión de varias habitaciones (reservadas por un mismo cliente para un mismo período) en una sola factura.

El *Sistema de Gestión de Hoteles* puede ser utilizado por cuatro tipos de usuarios:

- Usuario anónimo. Cualquier usuario que no se identifica explícitamente en la aplicación entra dentro de esta categoría. Un usuario anónimo puede consultar información acerca del hotel, sus instalaciones, las habitaciones de que dispone, los servicios que ofrece y los precios.
- Cliente. Un cliente puede darse de alta en el sistema. Además, tras haberse identificado en el sistema, un cliente puede realizar reservas a través de la web del hotel y tener acceso en cualquier momento a los datos de su(s) factura(s).
- Recepcionista. Las labores del recepcionista incluyen la consulta y realización de reservas por cuenta de un cliente y el mantenimiento de los cargos asociados a cada habitación. Si la reserva

implica la introducción de un cliente nuevo en el hotel, el recepcionista también está autorizado a darlo de alta. Un recepcionista también puede modificar los datos personales del cliente. Además, el recepcionista es el encargado de emitir facturas y gestionar el pago de las mismas.

- Administrador: además de poder realizar todas las labores de un recepcionista, el administrador es el encargado de realizar el mantenimiento de los datos generales del hotel, tales como los tipos de servicio ofertados, los tipos de habitación y sus precios o los métodos de pago permitidos.

Por claridad, a partir de este momento vamos a centrarnos en el modelado de la vista de la interfaz correspondiente al rol *recepcionista*. A continuación ilustraremos cómo OO-H puede ser aplicado a esta descripción para la consecución de una interfaz operativa que dé respuesta a los requisitos mencionados para este rol.

4.2 Análisis de Requisitos Funcionales

Como ya comentamos en el capítulo anterior, OO-H propone, siguiendo la estela de diversos procesos de modelado de software, la utilización de un *Diagrama de Casos de Uso (DCU)* con semántica y notación UML para la recogida de requisitos funcionales del sistema. Este diagrama está en la base del método, y dirige el resto de fases del proceso de construcción de la aplicación en base a los distintos tipos de usuarios (roles) identificados en ella.

4.2.1 Elementos de Modelado

Los principales elementos de modelado de este diagrama, definidos como parte de la especificación UML, son los *actores* y los *casos de uso*. Los actores pueden estar relacionados con los casos de uso mediante asociaciones, que indican que existe una comunicación entre una instancia del caso de uso y el usuario que ostenta

el rol representado por el actor. Los casos de uso pueden estar relacionados con otros casos de uso mediante relaciones de tipo inclusión (<<*include*>>), extensión (<<*extend*>>) y generalización (UML 1.4, 2001).

4.2.2 Método

Con el fin de construir el modelo de casos de uso de la aplicación web, OO-H sugiere los siguientes pasos (Rosenberg & Scott, 1998):

- Identificar actores.
- Para cada actor, identificar las actividades que éste realiza en el sistema.
- Agrupar las actividades en casos de uso.
- Establecer las asociaciones entre casos de uso y actores.
- Establecer las generalizaciones entre actores necesarias para simplificar el diagrama.
- Establecer las relaciones de tipo <<*include*>> y <<*extend*>> entre casos de uso.

Aunque en UML la especificación capturada mediante el Diagrama de Casos de Uso se puede enriquecer con diversos mecanismos (escenarios, diagramas de actividad, diagramas de colaboración o secuencia, etc.), OO-H no realiza ninguna recomendación respecto a su uso y, en caso de producirse, estos diagramas sólo son tenidos en cuenta como documentación del sistema, y no influyen en el resto de fases del método.

4.2.3 Aplicación al Caso de Estudio

Volviendo a nuestro ejemplo, y a modo de resumen, el *Sistema de Gestión de Hoteles* debe ofertar la siguiente funcionalidad para el rol *recepcionista*:

- Cualquier servicio ofrecido a un usuario anónimo, que en nuestro ejemplo se limita a ver los datos del propio hotel. Este requisito no exige que el usuario se identifique previamente en el sistema.

.

- Vista de las facturas de los clientes.
- Gestión de los cargos asociados a cada reserva.
- Capacidad de gestión de pago de facturas, que incluye la visualización de las mismas.
- Opción de visualización del estado de las reservas del hotel.
- Realización de nuevas reservas. Si esa reserva involucra a un nuevo cliente, posibilidad de alta de dicho cliente.
- Acceso a los datos de los clientes para su actualización.

Una vista parcial del diagrama de casos de uso correspondiente al actor *Recepcionista* y que incluye estos requisitos de usuario puede ser observada en la Fig. 4.1[1].

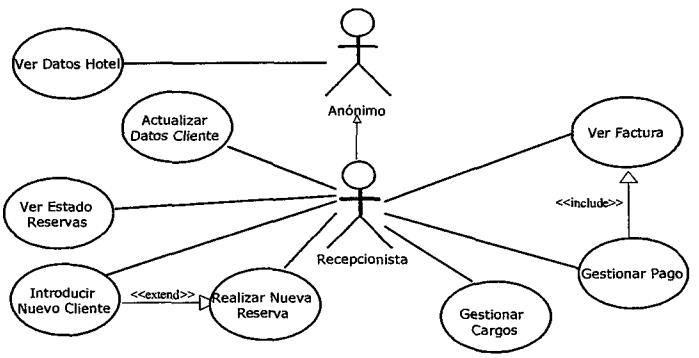

Figura 4.1. Vista parcial del DCU correspondiente al SGH

[1] En esta vista parcial, por motivos de simplicidad, se han obviado tanto los requisitos relativos al cliente y al administrador como las relaciones de inclusión o extensión que los casos de uso del recepcionista tienen con los del resto de actores

4.3 Análisis de Dominio

Una vez identificados los actores y la funcionalidad requerida, y capturados éstos en el diagrama de Casos de Uso, el siguiente paso consiste en construir un *Diagrama de Clases (DC)*, con notación y semántica UML, que organiza y encapsula tanto la estructura como la funcionalidad requerida de los conceptos relevantes de la aplicación, y que refleja la parte estática del sistema. En este momento aún no se tienen en cuenta cuestiones relativas a la navegación, presentación, flujo de datos o interacción, lo que simplifica y dota de una mayor capacidad de reuso al modelo.

4.3.1 Elementos de Modelado

Los elementos de modelado principales, presentes en el diagrama de clases, son las *clases* (con sus atributos y operaciones) y sus *relaciones* (asociación, agregación, composición, generalización). La semántica y representación gráfica de estos elementos coincide con la presentada en el contexto de la notación UML.

4.3.2 Método

Debido a que este análisis de dominio se materializa en un Diagrama de Clases UML estándar (ver Fig. 4.2), su construcción sigue técnicas de modelado orientadas a objetos bien conocidas, como son:

- Identificar clases.
- Determinar las asociaciones entre clases.
- Definir jerarquías de herencia.
- Especificar los atributos y las interfaces de los métodos relevantes.
- Identificar cardinalidades.

- Incluir como clases los actores identificados en la fase de análisis de requisitos funcionales de los que es necesario almacenar algún tipo de información[2].

En esta identificación de clases y relaciones es importante destacar que, una vez más, OO-H opta por simplificar al máximo la tarea, y por tanto se centra en las clases persistentes (*entity classes* (L.A.Maciaszek, 2001)), dejando de lado las clases de frontera y las clases de control presentes en otros modelos de análisis como es el Análisis de Robustez de Rosenberg (Rosenberg & Scott, 1998).

4.3.3 Aplicación al Caso de Estudio

Retomando el *Sistema de Gestión de Hoteles*, un posible modelo conceptual de análisis puede ser visto en la Fig. 4.2. En él observamos cómo se ha identificado una clase *Hotel* que tiene asociado, además de un nombre, una descripción y una categoría, un conjunto de *Instalaciones* de las que se pueden beneficiar sus clientes: piscina, restaurante, salas de reuniones, etc. El hotel también dispone de una serie de *Habitaciones*, cada una con una lista de reservas (clase *Reserva*) asociada. El sistema da la posibilidad de *crear* reservas mediante el método homónimo.

Además, cada reserva tiene asociada una lista de *Cargos*, correspondientes a los distintos *Servicios* solicitados por el cliente correspondiente. Cuando el cliente abandona el hotel, se le emite una *Factura*, que puede ser pagada con cualquiera de los *Medios de Pago* aceptados por el hotel.

En este sistema, el estereotipo <<*actor*>>[3] denota las clases que representan a los actores del sistema (excepto el usuario anónimo)

[2] Precisamente este último punto refleja una dicotomía interesante: un usuario del sistema representa tanto un objeto externo que interacciona con el mismo (y como tal se recoge en el diagrama de Casos de Uso) como una clase interna que recoge la información que permite identificarlo y caracterizarlo.

[3] Estereotipos, valores etiquetados y restricciones son los tres mecanismos de extensión estándares de UML (UML 1.4, 2001).

identificados durante la fase de análisis de requisitos funcionales (ver sección 4.2)[4].

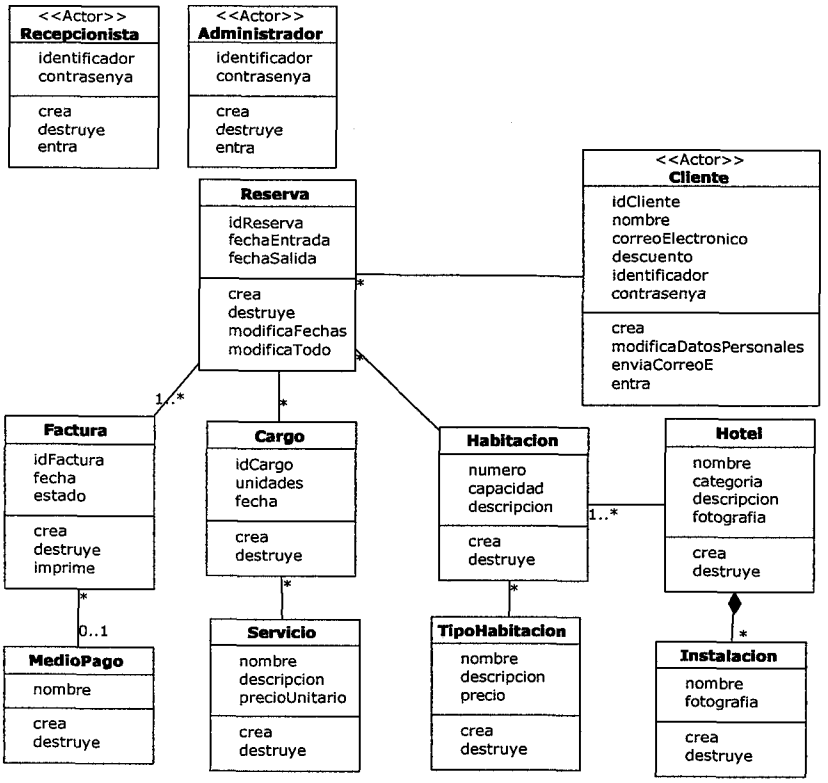

Figura 4.2. Diagrama de Clases de Análisis

Por último, todas las clases tienen un atributo que actúa de identificador externo de los objetos. Este atributo tiene asociado el valor etiquetado *key=true*, y en OO-H se marca visualmente mediante un icono que representa una llave (ver Fig. 4.2).

[4] Aunque la preservación de la simplicidad de los diagramas hace conveniente la ocultación de la jerarquía de herencia de usuarios, estos actores son en realidad en OO-H especializaciones de la clase genérica *Usuario*, que pertenece al marco de personalización de OO-H y que discutiremos en profundidad en el capítulo 7.

4.4 Diseño de Dominio

En OO-H el Diseño de Dominio es una continuación natural de la etapa de Análisis de Dominio presentada en la sección anterior, y su objetivo es doble. Por un lado en esta fase el modelo de análisis de dominio es refinado con características de un menor nivel de abstracción, como son los tipos de atributos o los parámetros de los métodos. Por otro, y dado que OO-H se ocupa de modelar la interfaz de la aplicación y no la aplicación completa, el diagrama de clases debe ser modificado para capturar, en caso de ser necesario, las interfaces a módulos[5] externos que queramos integrar en la aplicación.

A continuación veremos cada uno de estos aspectos.

4.4.1 Evolución del Modelo de Análisis de Dominio

Si durante la fase de análisis capturamos los conceptos básicos de la aplicación, la preocupación principal durante el diseño es el refinamiento de estos conceptos, y para ello se debe estudiar, entre otros aspectos:

- refinamiento de atributos: tipos.
- refinamiento de operaciones: revisión de la asignación de responsabilidades, precondiciones, postcondiciones, invariantes[6], parámetros y sus tipos, etc.
- atributos y métodos derivados.
- Aplicación de patrones de diseño que mejoren la solución final.

Elementos de Modelado. Los elementos de modelado son los mismos que los presentados para el modelo de análisis de dominio: *clases* (con sus atributos y operaciones) y *relaciones* (asociación, agregación, composición, generalización).

[5] Con este concepto nos referimos al tipo de componente que podría aparecer en un diagrama de componentes de UML.

[6] Precondiciones, postcondiciones e invariantes se definen en UML mediante el uso de OCL, que forma parte de la propia especificación UML (UML 1.4, 2001).

Método. La estructura del modelo de diseño de dominio está basada en la estructura del modelo de análisis, y consta de los siguientes pasos[7]:

- Revisar responsabilidades de las clases.
- Completar atributos y sus tipos.
- Completar tipo, carácter, valor por defecto, etc. de los parámetros de cada método.
- Refinar relaciones de agregación/composición.
- Detectar precondiciones, postcondiciones e invariantes del modelo.
- Detectar atributos y relaciones derivadas relevantes para la aplicación.
- Detectar tipos enumerados.
- Contemplar el uso de patrones de diseño para mejorar la estructura global de la aplicación.

Para la ejecución de esta fase la documentación preexistente del sistema (si es que existe) puede sernos de gran ayuda.

Aplicación al Caso de Estudio. Volviendo a nuestro caso de estudio, el resultado de esta actividad puede ser visto en la Fig. 4.3.

En ella se observa cómo la revisión de responsabilidades de las clases ha detectado nuevas necesidades, como es la inclusión de la operación *creaCjtoReservas* como parte de la clase *Reserva*, que permite que un cliente reserve varias habitaciones mediante una sola invocación. También hemos revisado las relaciones y hemos convertido las asociaciones entre la clase *Hotel* y las clases *Instalacion* y *Habitacion* en composiciones, con el fin de capturar la

[7] En general no existe acuerdo en cuanto a dónde está la frontera entre las actividades de análisis y las actividades de diseño de dominio. La división de actividades realizada en este trabajo es por tanto una recomendación y no una regla que deba ser seguida de manera estricta.

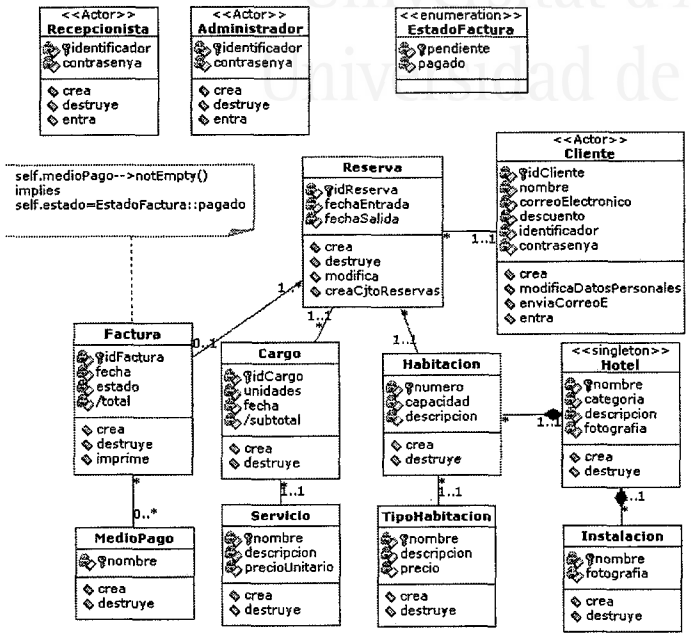

Figura 4.3. Diagrama de Clases de Diseño

semántica de pertenencia estricta de las instalaciones y habitaciones al hotel[8]. A la habitación le hemos añadido la responsabilidad de reservarse, posibilitando de este modo la creación de la relación entre *Reserva* y *Habitación* desde los dos extremos de la misma. Además, hemos incluido un servicio *existeReserva* que nos devolverá el valor *true* si la habitación está ocupada para un determinado período de tiempo. Esta operación es un ejemplo del tipo de operación cuyo valor etiquetado *isQuery=true* (definido

[8] Debido a que la relación de composición en el seno de la comunidad UML es causa de gran controversia (L.A.Maciaszek, 2001), creemos conveniente aclarar que, en el seno de OO-H, este tipo de relación implica que la clase componente no puede existir sin estar asociada a una clase (y sólo una) compuesta. Además, en OO-H consideramos que las clases componentes no puede cambiar de contenedor.

en el estándar UML[9]) le permite ser utilizada en fórmulas OCL asociadas a los distintos modelos.

Otro refinamiento ha sido la inclusión de un invariante (fórmula OCL) sobre la clase Factura que indica que, para que una factura tenga asociado un *Medio de Pago*, su estado debe ser igual a *pagado*. Para definir este estado hemos tenido que incluir el tipo enumerado *EstadoFactura*. Como podemos ver en la Fig. 4.3, un tipo enumerado se modela mediante el estereotipo <<*enumeration*>> aplicado sobre una clase, donde sus atributos representan los posibles valores que acepta. A la *Factura* también le hemos añadido un atributo derivado (cuya notación UML es una barra inclinada (/) delante del nombre) llamado *total* que almacena el montante total de la factura. Del mismo modo, hemos añadido un atributo derivado *subtotal* a cada *Cargo* realizado sobre una reserva. El valor de estos atributos se expresa nuevamente mediante una fórmula OCL (oculta en la Fig. 4.3) que se asocia a este tipo de atributos. Además, los atributos derivados tienen asociado un valor etiquetado que define si deben ser evaluados una sola vez (al crearse el objeto, como es el caso de los atributos *total* y *subtotal* de nuestro Sistema de Gestión de Hoteles), periódicamente (a intervalos de tiempo decididos por el diseñador) o cada vez que se visualiza su valor.

Por último, el ejemplo muestra la aplicación de un patrón de diseño: el patrón de *singleton* (Gamma *et al.*, 1995), que, asociado a la clase *Hotel* (estereotipo <<*singleton*>>), obliga a que en cada momento exista como mucho una instancia de la clase sobre

[9] El atributo isQuery se asigna en UML a los rasgos de comportamiento, como es el caso de las operaciones sobre las clases. Este atributo especifica si la ejecución de ese rasgo no altera el estado del sistema. Un valor de *cierto* indica que el estado no varía. Un valor establecido a *falso* implica por el contrario que la ejecución de ese rasgo puede causar efectos secundarios. La definición de operaciones de tipo *isQuery=true* permite su inclusión en fórmulas OCL (cuya ejecución por definición no puede hacer variar el estado del sistema) y por tanto no sólo simplifica la construcción de fórmulas OCL sino que aumenta la expresividad de dichas fórmulas, que pueden hacer referencia a operaciones genéricas incluidas como parte del modelo de dominio o incluso a circunstancias externas a la aplicación, consultadas a través de operaciones proporcionadas como parte del propio modelo de referencia de OO-H, tal y como veremos en el capítulo 7.

la que se aplica e indica por tanto que el sistema se centra en la gestión de un solo hotel.

4.4.2 Captura de interfaces de módulos preexistentes

Por otro lado, como ya hemos comentado, OO-H prevé la posibilidad de que existan módulos de lógica preexistentes que deban ser invocados desde la interfaz de aplicación que estemos modelando. Para ello, la interfaz de dichos módulos es integrada en el modelo de dominio de OO-H durante esta etapa de diseño[10].

El objetivo final es minimizar la interacción de conceptos contenidos en componentes de implementación distintos, y así mejorar aspectos de rendimiento, seguridad y acceso. El modo en que OO-H define estas vistas dentro de un marco orientado al objeto coherente es mediante los *Subsistemas* (mecanismo proporcionado por el propio UML) y mediante las *Clases de Ayuda*, utilizadas para subdividir entidades en unidades más simples que agrupan elementos relacionados por algún criterio (Rosenberg & Scott, 1998). A una misma clase (concepto) a nivel de diagrama de clases se le pueden asociar, mediante mecanismos de agregación, distintas clases que implementan distintas facetas de ese concepto.

Estas actividades hacen de OO-H una aproximación *top-down-up* (Retschitzegger & Schwinger, 2000) ya que, por un lado, se analizan los requisitos de usuario para, a partir de ellos, identificar los conceptos relevantes de dominio (visión top-down), y por otro se estudian las restricciones impuestas por la arquitectura de ejecución preexistente (si es que existen) para refinar el modelo de dominio obtenido en función de ellas (visión bottom-up). Desde OO-H pensamos que este modo de trabajar ayuda al diseñador en la toma de decisiones que no penalicen el rendimiento global de la aplicación.

[10] Es importante no confundir esta actividad con la actividad de integración existente en muchos ciclos de desarrollo de software, y que se centra en la integración incremental de los módulos de software que van siendo generados como resultado de las distintas iteraciones del método. Además, conviene clarificar que OO-H se ocupa sólo de modelar la interacción con esos elementos externos, lo que obliga a que dichos elementos ofrezcan una interfaz para su invocación de manera remota.

Elementos de Modelado. Con el fin de permitir el modelado de módulos de lógica y/o datos preexistentes, y haciendo uso de los mecanismos de extensión disponibles en UML, OO-H introduce un estereotipo de tipo <<*legacy*>> (ver Fig. 4.4), cuyo icono es una rueda dentada (ver Fig. 4.5) y que puede ser aplicado sobre un atributo, un método, una clase o un subsistema. Este estereotipo lleva asociado una serie de parámetros, entre los que destacan el tipo y ubicación de la fuente externa que da soporte a la sección del modelo afectada por el estereotipo.

Método. La inclusión de módulos preexistentes en la especificación de la interfaz se realiza de la siguiente manera:

- Decidir a qué elementos del modelo de dominio afecta cada uno de los componentes que se desean integrar.
- Introducir en el diagrama de clases los elementos legados simples mediante el estereotipo <<*legacy*>>.
- Para estructuras complejas, definir una o más clases de ayuda por cada módulo preexistente. Si el módulo integrado es complejo, agrupar las clases introducidas en un subsistema.
- Definir relaciones de agregación entre las clases del dominio y cada una de las clases de ayuda que le dan soporte.

Aplicación al Caso de Estudio. Volviendo al *Sistema de Gestión de Hoteles*, supongamos que el método *Pagar* es un servicio ofertado por una entidad bancaria que proporciona acceso a una rutina de pago que comprueba la validez de la tarjeta, realiza la transacción de pago y envía la conformidad al hotel. Una vista parcial del diagrama de diseño de dominio donde se muestra la introducción de la clase de ayuda *Banco* puede ser observada en la Fig. 4.4. Esta clase de ayuda se define sobre la clase inicial *Factura* para desglosar las responsabilidades propias de las que ostentan sistemas externos.

La forma de modelado alternativa, consistente en la integración del método *paga* como parte de la clase *Factura* dentro del diagrama de clases de dominio, se ilustra en la Fig.4.5.

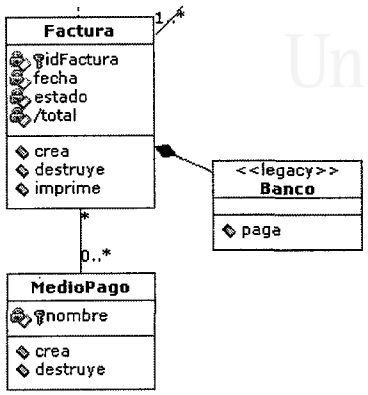

Figura 4.4. Elementos de Integración incluidos como Clases de Ayuda

Figura 4.5. Elementos de Integración embebidos en Clases de Dominio

4.5 Conclusiones del Capítulo

En este capítulo hemos presentado la fase de análisis y diseño de dominio de OO-H. El análisis de requisitos de usuario se materializa en un diagrama de Casos de Uso compatible UML, mientras que el análisis y diseño de dominio se refleja en un diagrama de clases, también compatible UML. El diagrama de clases de diseño puede ser refinado para reflejar la existencia de módulos legados que deben ser integrados con la interfaz de la aplicación que pretendemos modelar. Para ello OO-H incluye un estereotipo *<<legacy>>* y propone el uso de las clases de ayuda, con el fin de reflejar el carácter externo de los módulos y facilitar la toma de decisiones de diseño de interfaz que no penalicen el rendimiento global de la aplicación. Los diagramas resultantes, tal y como vimos en el capítulo 3, sirven como entrada de la etapa de diseño de navegación, que presentamos en los capítulos 5, 6 y 8.

5. El modelo de Navegación en OO-H

Todo debería simplificarse tanto como sea posible, pero no más.

ALBERT EINSTEIN

El capítulo 4 presentó el conjunto de actividades que permite modelar las tareas y conceptos de dominio relevantes para el desarrollo de una interfaz de calidad. Los diagramas resultantes se utilizan en OO-H como entrada de las actividades de análisis y diseño de navegación, contribución central del presente trabajo.

El objetivo principal de estas actividades es obtener un modelo que represente los caminos de navegación que se encuentran disponibles para resolver los requisitos planteados por cada tipo de usuario.

Antes de pasar a definir los conceptos que permiten la definición de este modelo de navegación, queremos clarificar que, pese a la similitud que el lector sin duda encontrará entre la notación utilizada por OO-H para la especificación de este modelo y la que define UML para la especificación de sus modelos estándares, **ninguno de los diagramas de navegación de OO-H es un diagrama UML**[1], ni la notación sigue las mismas reglas sintácticas ni semánticas. Por ejemplo, la notación utilizada para definir una *clase de navegación* es similar a la que se utiliza en UML para definir un objeto. Sin embargo, en OO-H esta notación (tal y como veremos en la Fig. 5.2) especifica una vista restringida sobre una clase de dominio, y por tanto sigue haciendo referencia a un contenedor.

[1] La definición de una propuesta compatible UML de estos diagramas queda como trabajo futuro.

Otro tema importante es el uso que OO-H hace de OCL durante las actividades de análisis y diseño de navegación para la definición de *filtros* que, como veremos en la sección 5.3.5, restringen tanto los caminos de navegación como los objetos de dominio que son visualizados como resultado de dicha navegación. En este caso OO-H ha optado por el uso de OCL para definir estas restricciones para evitar que el diseñador tenga que aprender un nuevo lenguaje de especificación de restricciones, distinto al que debe usar para especificar invariantes, precondiciones y/o postcondiciones sobre el diagrama de clases de la aplicación.

Sin embargo, el uso que OO-H hace de OCL a este nivel tampoco es estándar. En primer lugar, los filtros de navegación se aplican sobre los *enlaces navegacionales* (mientras que en UML no es posible aplicar restricciones OCL sobre las asociaciones). Además, los filtros de navegación en destino de OO-H permiten seleccionar colecciones de objetos, y no sólo evaluar restricciones sobre dichas colecciones. Por último, OO-H ha incluido nuevos símbolos. El primero de estos símbolos es el signo de interrogación (*?*), que permite definir un valor que debe ser introducido en tiempo de ejecución. Ese valor puede estar circunscrito a un conjunto de valores posibles, restricción que se expresa mediante el operador *valueIn*. Por último, OO-H permite el uso del operador *like* y el uso del carácter comodín * para la comparación de cadenas. Todas estas extensiones son soportadas en el compilador OCL incluido como parte del entorno de desarrollo de OO-H, que presentaremos en el capítulo 9.

Una vez clarificado este punto, en la sección 5.1 definiremos los dos niveles (análisis y diseño) en los que OO-H divide la navegación de un sistema hipermedial. A continuación, en la sección 5.2 presentaremos la actividad de análisis de navegación, que genera como resultado un *Diagrama de Navegación Semántica (DNS)*. Sobre este diagrama, y con la ayuda del modelo obtenido como resultado de las actividades de análisis y diseño de dominio presentadas en el capítulo 4, la sección 5.3 presenta los refinamientos que introduce la actividad de diseño de navegación de OO-H. Esta actividad genera como resultado un *Diagrama de Acceso Navega-*

cional (DAN). Este diagrama recoge las decisiones acerca tanto de los caminos a través de la información del dominio como de las interfaces que deben ser proporcionadas para interaccionar con servicios de lógica preexistentes (aspecto este último que será tratado en profundidad en el capítulo 6). En la sección 5.4 profundizaremos en la forma en que la información recogida en el DNS influye y dirige la estructura genérica del DAN. Por último, la sección 5.5 presenta las conclusiones del capítulo.

5.1 El concepto de Navegación en OO-H

Como ya comentamos en el capítulo 3, y de acuerdo con la Filosofía del Diseño Hipermedial (Yoo & Bieber, 2000), la existencia y rigurosidad de un análisis y diseño de navegación es importante para cualquier tipo de modelo de interfaz, y no sólo para las interfaces hipermediales.

Sin embargo, su especial relevancia dentro del proceso de diseño de aplicaciones web ha provocado que todos los métodos de modelado hipermedial, hasta lo que alcanza nuestro conocimiento, hayan incluido un modelo de diseño navegacional en su proceso de desarrollo, tal y como vimos en el capítulo 2. Estos modelos de diseño son en la mayoría de los casos (quizás la excepción más notable sea la de WAE (Conallen, 1999)) independientes del lenguaje de implementación final, pero no ocurre lo mismo con respecto a la arquitectura de ejecución, el modelo de interacción de usuario, la plataforma o el dispositivo de acceso que el diseñador tenga en mente. El motivo es que todos ellos asumen explícita o implícitamente una determinada estructura de las páginas abstractas que componen la aplicación (*siteview* del sistema), que claramente es dependiente de los aspectos mencionados.

Este bajo nivel de abstracción se convierte en un problema ante la creciente necesidad de implementar la misma aplicación para distintos entornos. En estas ocasiones, no sólo hay que modificar el modelo de navegación al más alto nivel, sino que el hecho de basarse en el mismo modelo de dominio y los mismos requisitos no

asegura que las distintas implementaciones conserven una coherencia a nivel de estructuración de interfaz. Aún más, los modelos de diseño de navegación actuales suelen relacionarse fuertemente con el modelo de dominio subyacente. Este hecho introduce el riesgo de que la estructura global de la interfaz (y no sólo los caminos de navegación que materializan las relaciones entre objetos) sea dirigida implícitamente por la estructuración de los datos, y no por las necesidades del usuario, lo cual menoscaba su facilidad de uso (Nunes & e Cunya, 2000).

Es por ello que OO-H define dos tipos de navegación, según el nivel de abstracción al que nos encontremos (Cachero & Koch, 2002; Cachero *et al.*, 2002a; Cachero *et al.*, 2002b). A nivel de análisis, OO-H define el concepto de *Navegación Semántica*, y plantea una estructura de interfaz de usuario de alto nivel en base a requisitos de usuario. A nivel de diseño, OO-H introduce el concepto de *Navegación Estructural*, más específico, que se ocupa de aspectos como estructuras de acceso a la información, saltos entre vistas de usuario o caminos de invocación de servicios. A continuación veremos en profundidad cada uno de estos conceptos.

5.1.1 El concepto de Navegación Semántica

Definimos el concepto de *Navegación Semántica* como *la consecuencia de un cambio en la intención del usuario* o, dicho de otro modo, como la activación por parte del usuario de un enlace que da acceso a un nuevo requisito funcional.

Esta definición tiene dos implicaciones importantes: la primera es que, desde el punto de vista de OO-H, el usuario es el único que puede navegar, lo que a su vez supone que cualquier cambio de requisito activado por la propia aplicación no implica navegación. La segunda implicación es que, debido al hecho de que son los saltos entre requisitos de usuario los que determinan esta navegación semántica, el nivel de granularidad al que se definan estos requisitos determina qué es o no es navegación. Por tanto el concepto de navegación semántica incrementa de manera substancial el peso

de un correcto proceso de definición de requisitos en la calidad de la interfaz final.

5.1.2 El concepto de Navegación Estructural

A nivel de diseño de interfaz, definimos la *Navegación Estructural* como una *acción voluntaria de usuario que causa un cambio en su vista de interfaz* (y no en su intención, como ocurría con la navegación semántica). La mayoría de estructuras navegacionales incluidas como parte de las propuestas hipermediales estudiadas en el capítulo 2 se centran en este tipo de navegación.

Estos dos conceptos dan lugar en OO-H a dos actividades, tal y como vimos en el capítulo 3: análisis de navegación y diseño de navegación. El resultado de la combinación de ambas actividades es un modelo de navegación que permite la generación automática de prototipos, y cuya semántica y notación presentamos a continuación.

5.2 Análisis de Navegación

OO-H coincide con metodologías como Wisdom (Nunes & e Cunya, 2000) y defiende la estructuración explícita de la interfaz de usuario en base a tareas (Paterno & Mancini, 1999) con el fin de evitar el riesgo de que su organización acabe siendo dirigida implícitamente por la estructuración de los datos subyacentes, menos intuitiva para el usuario final. En esta línea, el análisis de navegación en OO-H es una actividad independiente del análisis de dominio, y su objetivo es definir una estructura de interfaz que permita acceder al modo en que la aplicación da respuesta a cada uno de los requisitos funcionales, independientemente de modelo de interacción, plataforma y/o dispositivo de acceso.

El *Diagrama de Navegación Semántica (DNS)* generado como resultado de esta actividad no sólo permite la reflexión sobre el modo en que se prevé que el usuario utilice la aplicación sino que

además, al igual que el modelo de Casos de Uso o el modelo de Análisis de Dominio, es reutilizable entre implementaciones.

Una consecuencia de este hecho es la reducción del coste de cambio (en términos de facilidad de uso) entre estas implementaciones de la misma aplicación, ya que su uso garantiza una estructura global común. Además, el DNS constituye un puente entre el modelo de requisitos funcionales y el modelo de diseño de navegación, y aumenta por tanto la trazabilidad del método, posibilitando además la comprobación automática de que todos los requisitos funcionales están cubiertos en el modelo de navegación del sistema.

5.2.1 Elementos de Modelado

Los principales elementos de modelado del análisis de navegación son los *Enlaces Semánticos* y las *Unidades Semánticas*.

Enlaces Semánticos. Los *Enlaces Semánticos* son constructores que proporcionan al usuario un modo de acceder a la respuesta que la aplicación ofrece a cada requisito funcional. Su principal característica es la estabilidad frente a cambios de plataforma, dispositivo de acceso o modelo de interacción, y su representación gráfica es una flecha dirigida y etiquetada con el nombre del requisito destino del salto navegacional (ver Fig. 5.1).

Los enlaces semánticos tienen como destino (y, opcionalmente, también como origen), una unidad semántica, cuya definición presentamos a continuación.

Unidades Semánticas. Una *Unidad Semántica* es un constructor que representa al conjunto de estructuras de información y navegación que colaboran en dar respuesta a un subconjunto de requisitos funcionales relacionados. Los criterios de esta agrupación serán vistos en la sección 5.2.2. Su representación gráfica es el símbolo de *paquete* de UML, tal y como veremos cuando presentemos el DNS correspondiente a nuestro caso de estudio en la Fig. 5.1.

5.2.2 Método

Una vez definidos los principales constructores, el análisis de navegación en OO-H se realiza mediante los siguientes pasos:

- Agrupar los casos de uso atendiendo a criterios de uso de la aplicación.
- Definir un enlace semántico por cada requisito funcional identificado en la fase de análisis de requisitos. Este enlace tendrá como destino la unidad semántica en la que se haya englobado el correspondiente requisito. Para definir el origen se debe analizar si, desde el punto de vista del usuario, es conveniente el acceso a ese requisito directamente desde la entrada de la aplicación. Si es así, definir un enlace semántico sin origen. Si no, definir como origen la unidad semántica donde se da respuesta a algún requisito desde el cual es posible cambiar al requisito actual.
- Si es necesario, enriquecer el modelo con enlaces semánticos redundantes (que representen a un requisito que ya había sido cubierto por algún otro enlace). De este modo se aumenta la conectividad de la aplicación.

Criterios de Agrupación de Requisitos Funcionales. Con el fin de agrupar los casos de uso bajo una misma unidad semántica, se pueden aplicar diversos criterios, en función de sus previsiones de uso. Estos criterios son los siguientes:

- Similitud de Propósito. Algunos requisitos comparten un objetivo general. Como ejemplo, podemos imaginar una unidad semántica que agrupe todos los informes que pueden ser generados por la aplicación.
- Dependencia Funcional de Vistas. Otros requisitos se acceden siempre en conjunción con (como soporte de) otros. Como ejemplo, en nuestro hotel la creación de un cliente se realiza normalmente como consecuencia de un deseo de efectuar una reserva para ese cliente, por lo que podríamos decidir agrupar ambos requisitos bajo una misma unidad semántica.

- Estructura de datos. Otros requisitos operan con los mismos conceptos de dominio. Por ejemplo, los requisitos añadir, modificar y borrar un cliente del hotel trabajan todos con el concepto *Cliente*, por lo que podrían ser agrupados.
- Estado del objeto. Algunos requisitos se activan cuando un determinado objeto se encuentra en un determinado estado. Por ejemplo, la modificación y baja de servicios, habitaciones o tipos de habitación de nuestro hotel se debe realizar cuando la aplicación se encuentre en estado inactivo, por lo que todas estas operaciones de mantenimiento se podrían agrupar bajo una misma unidad semántica. En este punto es importante hacer notar que el cambio de estado de un objeto puede estar causado por la ocurrencia de un evento de la propia aplicación (e.g. arrancar la aplicación) o por la ocurrencia de un evento temporal.

Estos criterios normalmente se combinan en el ámbito de la misma aplicación, y la elección de agrupación de casos de uso teniendo en cuenta uno u otro normalmente depende de las previsiones de uso del sistema.

5.2.3 Aplicación al caso de estudio

Partiendo del Diagrama de Casos de Uso de la Fig. 4.1, una posible agrupación de los requisitos identificados en cuatro unidades semánticas (ver Fig. 5.1) sería la siguiente:

- Reservas: gestión de la ocupación del hotel. Agrupa a los Casos de Uso *Realizar Nueva Reserva* y *Ver Ocupación del Hotel*. En este caso el criterio seguido ha sido el de dependencia funcional de vistas: la vista sintetizada de la ocupación del hotel actúa de apoyo para la realización de nuevas reservas, ya que permite evitar intentos de reserva en días y/o sobre habitaciones en los que el hotel está ya completo.
- Facturas: gestión de pagos del hotel (tanto de la propia habitación como de los cargos asociados a ella). Agrupa a los Casos de Uso *Gestionar Cargos*, *Gestionar Pago* y *Ver Facturas*. Esta

agrupación es debida a la aplicación del criterio de estructura de datos común, ya que todos los requisitos trabajan con el concepto de *Factura*.

- Clientes: gestión de clientes del hotel. Agrupa a los Casos de Uso *Introducir Nuevo Cliente* y *Actualizar Datos Cliente*. Nuevamente esta agrupación ha sido realizada mediante la aplicación del criterio de estructura de datos común, ya que todos los requisitos se materializan en cambios sobre el concepto *Cliente*.
- Promocional: información general del hotel y emplazamiento turístico. Agrupa a los Casos de Uso *Ver Datos Hotel* y *Ver Información Turística*. Estos casos de uso tienen como objetivo común el proporcionar información promocional del hotel, y por tanto el criterio aplicado ha sido el de la similitud de propósito.

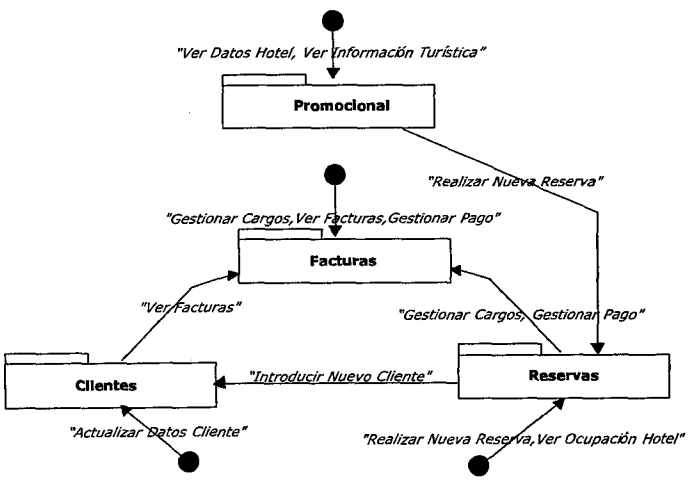

Figura 5.1. Diagrama de Análisis Navegacional del Sistema de Gestión de Hoteles

En la Fig. 5.1 también podemos observar cómo sobre estas unidades semánticas se han definido los enlaces semánticos. El modo de definirlos nos indica que el recepcionista sólo puede introducir un cliente nuevo en el sistema cuando detecta la necesidad de cambiar a este requisito mientras está gestionando reservas. También

podemos observar cómo está permitido pasar de un entorno de gestión de clientes a un entorno de gestión de facturas, sin pasar por el menú principal de la aplicación. Es importante resaltar que puede existir más de un enlace semántico asociado a un mismo requisito de usuario (ver e.g. el enlace semántico *Ver Facturas* en la Fig. 5.1).

Una vez definido el DNS, el siguiente paso del método consiste en la ejecución de la actividad de diseño de navegación, que detallamos a continuación.

5.3 El Diseño de Navegación

El diseño de navegación, tal y como vimos en el capítulo 2, es un elemento presente en todos los métodos de modelado hipermedial, y constituye uno de sus principales rasgos diferenciadores. El modelado de los caminos semánticamente relevantes a través del espacio de información no sólo documenta el sistema, sino que ayuda a controlar su complejidad y a evaluar la adecuación de la aplicación para responder a los requisitos de usuario.

OO-H captura estas decisiones de navegación (qué información se encuentra accesible para cada usuario, y cómo puede acceder a ella) en un conjunto de *Diagramas de Acceso Navegacional (DAN)* (Gómez, 2000), cada uno de los cuales está asociado a un actor del sistema[2].

5.3.1 Elementos de Modelado

Tal y como podemos ver en la Fig. 5.3, el DAN se construye a partir de cuatro constructores genéricos:

- Destino Navegacional (DN)

[2] Recordemos que los actores del sistema fueron identificados durante la fase de análisis de requisitos, y dieron lugar a un conjunto de clases estereotipadas como <<actor>> en el modelo de dominio.

- Colección (C)
- Clase Navegacional (CN)
- Enlace Navegacional (EN)

A continuación explicaremos cada uno de ellos.

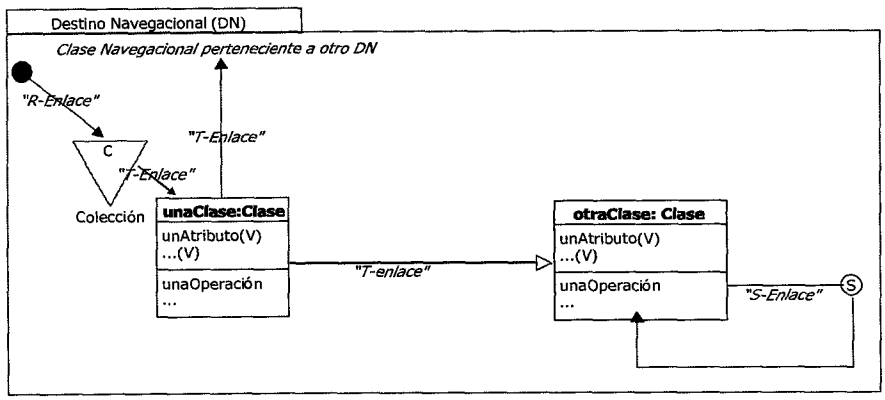

Figura 5.2. Iconos correspondientes a los constructores del DAN

5.3.2 *El Destino Navegacional*

Los Destinos Navegacionales (DN) agrupan los elementos del modelo que colaboran en la cobertura de los distintos requisitos navegacionales del usuario. Su símbolo es el *paquete* de UML, tal y como podemos ver en la Fig. 5.2. Los DN simplifican la definición de la estructura de la navegación al descomponer la complejidad de la tarea en distintos niveles de abstracción.

5.3.3 *La Colección*

Las Colecciones (C) son estructuras de agrupación (menús) de Enlaces Navegacionales (que presentaremos en la sección 5.3.5).

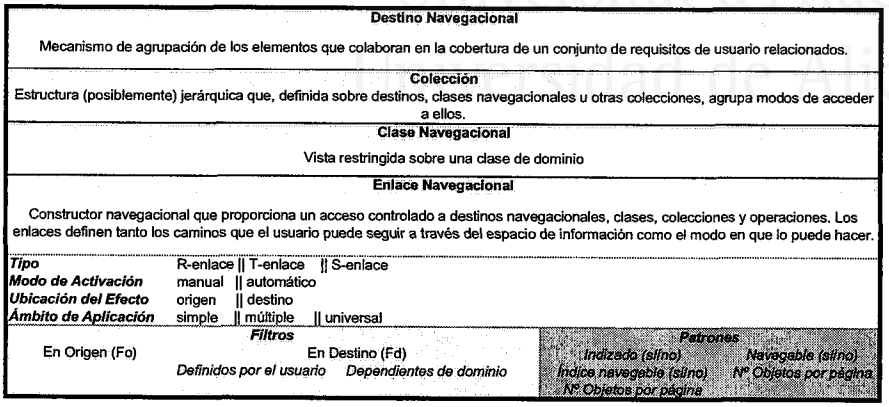

Figura 5.3. Principales constructores del DAN

Su símbolo es un triángulo invertido, tal y como puede observarse en la Fig. 5.2.

La estructura usual de agrupación de las Colecciones es la estructura lineal: un conjunto de opciones, cada una de las cuales conduce a una vista distinta de la aplicación. Como ejemplo, podemos diseñar un menú de primer nivel para nuestro hotel que contenga los enlaces necesarios para acceder a los subsistemas *Promocional*, *Reservas*, *Facturas* o *Clientes* que veíamos en la Fig. 5.1. Sin embargo, ésta no es la única estructura posible. Se pueden definir por ejemplo colecciones con una estructura interna jerárquica, es decir, que contienen una serie de menús anidados, donde sólo el último nivel corresponde a la activación de un enlace navegacional.

5.3.4 *La Clase Navegacional*

Las Clases Navegacionales (CN) son vistas parciales sobre clases de dominio, y reflejan el grado de interés que cada uno de sus atributos/operaciones tiene para cada tipo de usuario. Este concepto aparece con distintos nombres en las aproximaciones más significativas estudiadas en el capítulo 2. En OO-H, su icono es

un rectángulo con tres áreas, donde la superior muestra el nombre de la clase navegacional y la clase de dominio de la que deriva, y las dos inferiores muestran un subconjunto de los atributos y operaciones de esta clase de dominio, visibles a través de la clase navegacional. Este icono puede ser visto en la Fig. 5.2.

El filtrado de la estructura de información y funcionalidad relevante para cada usuario se realiza mediante la adición de una etiqueta a cada atributo/operación, que recoge su grado de disponibilidad (**Visibilidad**) para cada tipo usuario.

Este valor etiquetado puede tomar tres valores:

- Visible (V-Atributos): indica que el atributo/operación de la clase de dominio va a ser accesible a través de la clase navegacional. Es el valor por defecto.
- Referenciable (R-Atributos): este tipo de visibilidad, sólo aplicable a atributos, indica que dicho atributo proporciona una información que no es vital para el usuario, y por tanto se muestra sólo si el usuario lo solicita de manera expresa mediante un salto de navegación. De este modo se facilita la definición de vistas condensadas y extendidas de un mismo objeto.
- Oculto (O-Atributos): indica que el atributo/operación no es accesible desde la vista que se esté modelando en ese instante.

Los atributos/operaciones ocultos no aparecen en el diagrama, mientras que los atributos visibles o referenciables se distinguen gracias a un decorador (*(V)* o *(R)* respectivamente) que aparece detrás de su nombre, tal y como podemos ver en la Fig. 5.2.

5.3.5 *El Enlace Navegacional*

Una vez restringido el espacio de información del tipo de usuario considerado mediante la definición de clases navegacionales, el siguiente paso consiste en modelar los posibles movimientos del usuario a través de dicho espacio de información. Para ello OO-H introduce un nuevo constructor, denominado *Enlace Navegacional*

(EN). La semántica de los EN es una de las principales aportaciones de este trabajo, y permite la definición no sólo de los caminos a través de la información sino de los modos de interacción del usuario con la aplicación.

Para ello, OO-H distingue entre tres tipos de enlace, tal y como podemos observar en la Fig. 5.3:

- **R-Enlace (Enlaces de Requisito)**. Un R-Enlace marca el inicio de uno o varios caminos de navegación, destinados a dar respuesta a un subconjunto de requisitos funcionales. Los R-Enlaces se caracterizan por carecer de un origen explícito, y su representación gráfica puede ser vista en la Fig. 5.2. En caso de encontrarse en el seno de DN, se supone que el origen está establecido en un nivel superior. Si nos encontramos en el nivel superior del DAN, el origen se asume que se encuentra en algún lugar fuera de nuestro ámbito de modelado.
- **T-Enlace (Enlaces de Travesía)**[3]. Los T-Enlaces definen caminos de navegación entre objetos, ya sea dentro de un DN determinado, entre DN pertenecientes al mismo tipo de usuario o incluso entre DN pertenecientes a distintos diagramas de navegación (construidos para distintos tipos de usuario), permitiendo de este modo el reuso de partes del esquema navegacional entre tipos de usuario distintos. Su representación gráfica puede ser vista en la Fig. 5.2.
- **S-Enlace (Enlaces de Servicio)**. Los S-Enlaces, por último, son los de semántica más compleja, y definen el conjunto de caminos de navegación que permiten la interacción con módulos de lógica preexistentes. Este conjunto está compuesto en primer lugar por uno o más caminos que, al ser recorridos, permiten la recogida de valores de entrada al servicio. Una vez ejecutado el método subyacente, los S-Enlaces proporcionan además un

[3] En versiones anteriores del método se distinguía entre I-Enlaces (Enlaces Internos), circunscritos a un DN concreto, y T-Enlaces (Enlaces de Travesía) que se utilizaban cuando se deseaba interconectar DN. Sin embargo, la similitud semántica entre ellos nos ha llevado a considerar el I-Enlace como un caso particular de un T-Enlace donde el destino se encuentra dentro del mismo DN, y por tanto se ha eliminado como constructor independiente.

camino de vuelta que, al ser recorrido, muestra el resultado de la ejecución del servicio. Por último, el S-Enlace especifica por dónde puede seguir navegando el usuario una vez visualizados los resultados de dicha ejecución. La representación gráfica de los S-Enlaces, que serán tratados en profundidad en el capítulo 6, puede ser vista en la Fig. 5.2.

Atributos de los Enlaces Navegacionales. Independientemente de su tipo, los Enlaces Navegacionales en OO-H comparten una serie de atributos, que son:

- **Modo de activación**. Hablamos de enlaces *manuales* cuando es el usuario el que decide (mediante una pulsación de ratón o cualquier otro mecanismo) cambiar de página abstracta. Por el contrario, hablamos de enlaces *automáticos* cuando es la propia visualización del enlace la que implica su activación. Por defecto, el valor del modo de activación de los enlaces en OO-H es manual.
- **Ubicación del efecto**. La activación del enlace puede suponer el enriquecimiento de la página abstracta actual, en cuyo caso hablamos de una *visualización en origen*, o el salto físico a una nueva página abstracta, en cuyo caso nos encontramos ante una *visualización en destino*. En este sentido OO-H se distingue de otras aproximaciones, que realizan una composición previa de los conceptos de dominio que pueden aparecer en una misma página abstracta, y así eliminan la necesidad de este atributo de enlace. El motivo por el que OO-H no hace lo mismo es doble. Por un lado, el disponer de este tipo de enlace facilita el reuso de los modelos entre plataformas, siempre que la información que se desea visualizar sea la misma: basta cambiar el carácter en origen/destino de un enlace para aumentar o disminuir el tamaño de las páginas abstractas, sin necesidad de redefinir las clases navegacionales. Además, este atributo es útil para la materialización, en caso de ser necesario, de menús de enlaces físicos internos (de página), muy comunes en las aplicaciones hipermediales. Con el fin de diferenciar visualmente los enlaces en origen de los enlaces en destino, los primeros se repre-

sentan con una punta de flecha hueca, mientras que los enlaces en destino se representan con la punta rellena (ver Fig. 5.2). Por defecto, la ubicación del efecto es en destino.

- **Ámbito de Aplicación.** Por último, los enlaces contextuales (i.e. aquellos que trasladan consigo información acerca de su origen) pueden trasladar información de un solo objeto del origen (y entonces hablamos de ámbito de aplicación *simple*), de un conjunto de objetos elegidos por el usuario en tiempo de ejecución, (dando lugar a un ámbito de aplicación *múltiple*) o trasladar la información referente a todos los objetos que estaban siendo visualizados en el origen, en cuyo caso nos encontramos ante un ámbito de aplicación *universal*. El ámbito de aplicación múltiple tiene además un atributo adicional que especifica el número mínimo y/o máximo de instancias que deben ser seleccionadas en el origen. Por defecto, todos los enlaces tienen un ámbito de aplicación simple, salvo que se trate de enlaces de servicio asociados a métodos de clase, en cuyo caso el ámbito de aplicación está predefinido como universal.

Por último, todos los tipos de enlace pueden tener asociados tanto **Patrones de Navegación** como **Filtros de Navegación**.

Patrones de Navegación. Los *Patrones de Navegación* se definen en OO-H en base a dos características ortogonales (ver Fig. 5.3):

- Indizado (y, si su valor es cierto, si es navegable o no y parámetro de paginación[4] del índice)
- Navegación interna del conjunto de objetos (y, si su valor es cierto, parámetro de paginación del conjunto de objetos).

Instanciaciones particulares de estos valores dan lugar a los cuatro patrones de navegación más conocidos:

- Patrón Índice: supone una navegación indizada donde se presentan las referencias correspondientes a todos los objetos de la

[4] Número de objetos que deben aparecer en cada página, sea de índice o no.

colección en una sola página de índice, y donde no existe navegación interna entre objetos destino, por lo que el único modo de acceder a otro objeto de la colección es a través del índice. Los atributos que definen un índice son aquéllos que pertenecen a la clase destino del enlace o a cualquiera de las clases de navegación que cumplen dos condiciones: (1) estar relacionadas con ésta a nivel de DAN mediante un enlace en origen y (2) existir a nivel de Diagrama de Clases una relación estructural entre ellas con cardinalidad 1:1.

- Patrón Visita Guiada: supone una navegación no indizada en la que existe una navegación interna y donde los objetos afectados por esta navegación se presentan de uno en uno.
- Patrón Visita Guiada Indizada: implica una navegación indizada en el ámbito de un conjunto de objetos entre los cuales se ha definido una navegación de uno en uno. A diferencia del Patrón Índice, en este patrón el índice no selecciona un objeto de la colección sino que indica el objeto por el cual se empieza a navegar dicha colección.
- Patrón Mostrar Todo: implica una navegación sin indización previa y sin navegación interna, por lo que todos los objetos se muestran en una misma página abstracta. Es el patrón por defecto.

Una esquematización de estos patrones puede ser vista en las Figs. 5.4 a 5.7.

La flexibilidad de este mecanismo se hace patente si pensamos en índices arbitrarios, como es el caso de *los diez productos más vendidos*. El modelado de este tipo de índice, imposible en otras aproximaciones, se realiza en OO-H ordenando los productos por número de ventas y definiendo un patron de navegación de índice con un *parámetro de paginación=10* y *navegable=no*. Esto causa que sólo se muestre la primera página de índice, es decir, el índice de los diez más vendidos que estábamos buscando.

Filtros de Navegación. Los filtros de navegación por su parte son fórmulas OCL que se utilizan, de una manera no estándar,

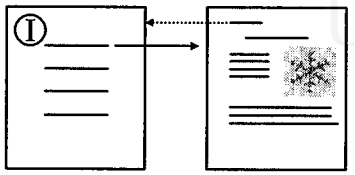

Figura 5.4. Patrón de Navegación Índice

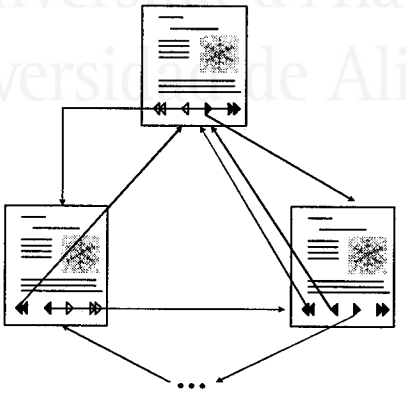

Figura 5.5. Patrón de Navegación Visita Guiada

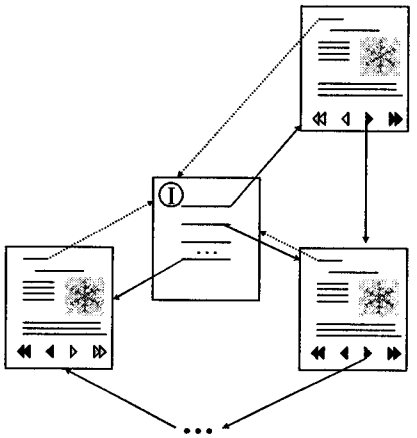

Figura 5.6. Patrón de Navegación Visita Guiada Indizada

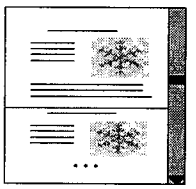

Figura 5.7. Patrón de Navegación Mostrar Todo

para especificar restricciones tanto sobre la disponibilidad de los caminos de navegación como sobre la población destino de los enlaces. En la Fig. 5.8 podemos ver algunos ejemplos de filtros definidos sobre los enlaces navegacionales *Ver Facturas* y *Factura Asociada.* A continuación veremos con detenimiento cada uno de ellos.

OO-H define dos tipos de filtros: *Filtros de origen (Fo)* y *Filtros de destino (Fd).* Los Fo restringen, como su nombre indica, el conjunto de objetos que pueden ser origen de la navegación representada por ese enlace. A modo de ejemplo, supongamos que queremos ver la lista de clientes del hotel y, para aquéllos que tienen facturas pendientes, dar la opción al usuario de ver sus facturas. En este caso el encargado de especificar que el enlace *Ver Facturas* sólo debe estar activo para aquellos clientes que tengan facturas pendientes es un Fo como el que presentamos en la Fig. 5.8.

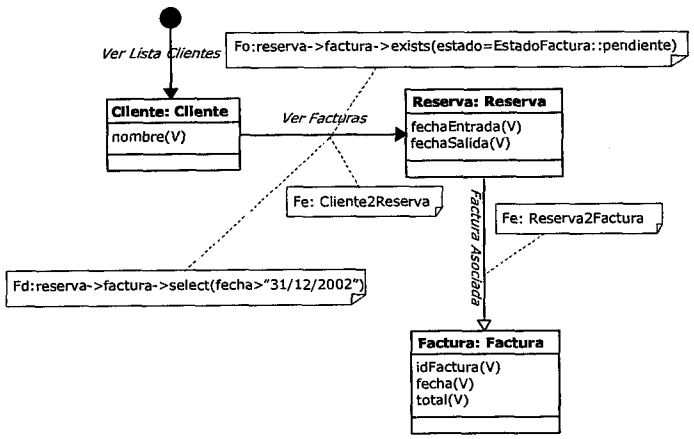

Figura 5.8. Filtros de origen (Fo) y destino (Fd) necesarios para mostrar las facturas del año actual de los clientes morosos

Los Fd, por su parte, restringen el conjunto de objetos destino obtenidos como fruto de esa navegación. Si refinamos el ejemplo anterior y especificamos que sólo queremos ver las facturas del

año 2003, es necesario añadir un Fd que establezca que, para que una factura pueda ser visualizada cuando navego por el enlace *Ver Facturas*, su *fecha*) tiene que ser mayor que el *31/12/2002* (ver Fig. 5.8).

Dentro de los Fd existe un caso especial, y es el de los *Filtros Estructurales (Fe)*. Un Fe es un Fd que, además, refleja una relación estructural (proveniente del Diagrama de Clases de Dominio). Cada vez que en OO-H se navega de una CN a otra, y siempre que entre ellas exista alguna relación estructural, se pone a disposición del diseñador el conjunto de Fe que corresponden a las relaciones existentes entre las dos clases. De este modo se simplifica su trabajo, al evitarle tener que introducir esas relaciones estructurales mediante fórmulas OCL. En este punto, es importante destacar la flexibilidad de este mecanismo, que permite reutilizar las relaciones de dominio para la navegación sin restringir el espectro de navegaciones posibles a esas relaciones, ya que el diseñador puede además introducir cualquier otro camino y asociarle el tipo de restricción que desee. Esta aproximación contrasta con la seguida en la mayoría de los métodos hipermediales estudiados en el capítulo 2, que limitan los caminos de navegación a aquéllos determinados por las relaciones de dominio. Volviendo a nuestro ejemplo, la activación de los Fe que permiten mostrar sólo las facturas de un determinado cliente (*Cliente2Reserva* y *Reserva2Factura*) puede ser vista en la Fig. 5.8.

5.3.6 Método

En OO-H este diseño navegacional se realiza de manera independiente para cada usuario, y consta de cuatro pasos:

- Estructuración de la interfaz en DN en función de las unidades semánticas definidas en la fase de análisis.
- Asignación de clases de dominio que proporcionan los datos y servicios que permiten dar respuesta a los requisitos funcionales cubiertos por cada DN.

- Agrupación de la información en páginas abstractas mediante la definición de enlaces en origen/destino
- Enriquecimiento de la semántica de los enlaces mediante la definición de filtros y la asignación de un ámbito de aplicación, un modo de activación y/o un patrón de navegación

5.3.7 Aplicación al Caso de Estudio

Siguiendo con el ejemplo del *Sistema de Gestión de Hoteles*, recordemos que durante la fase de análisis de requisitos decidimos definir cuatro USN: *Promocional*, *Clientes*, *Facturas* y *Reservas* (ver Fig. 5.1). En OO-H la definición de unidades semánticas induce la estructura navegacional de la interfaz, y por tanto da lugar a un DAN de nivel 1 compuesto por cuatro DN y un menú de acceso (colección), tal y como puede ser visto en la Fig. 5.9.

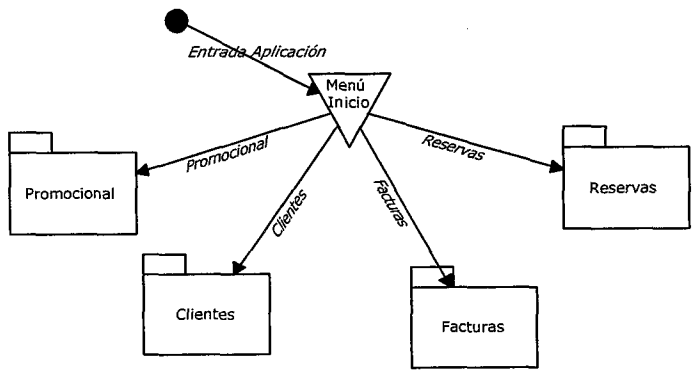

Figura 5.9. Primer Nivel de la Estructura Navegacional del SGH

Este primer nivel, debido a que surge directamente del modelo de análisis de navegación, es todavía independiente del dispositivo de acceso empleado para acceder a la aplicación.

A partir de ahí, vamos a suponer que deseamos modelar el requisito *Ver Ocupación Hotel*, contenido dentro del DN *Reservas*. Para ello vamos a suponer que el tipo de interfaz que queremos

modelar está compuesta por dos páginas abstractas. La primera página es una lista que contiene el nombre del cliente, la fecha de llegada, la fecha de partida y el número de habitación en la que se encuentra cada cliente alojado en el hotel. Además, desde esta vista, el recepcionista puede acceder a otra vista que le permite *Realizar Nueva Reserva*. Para ello, la aplicación pide una serie de parámetros (cuya forma de modelado será vista en el capítulo 6) y, una vez invocado el servicio, vuelve a la vista de reservas del hotel, donde ahora aparecerá la nueva reserva introducida. Este modelo de interacción usuario-sistema se plasma en OO-H en un diagrama de navegación de nivel 2 como el presentado en la Fig. 5.10.

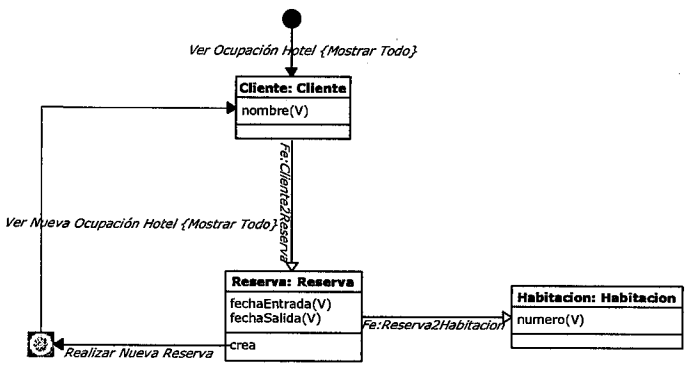

Figura 5.10. DAN de Segundo Nivel correspondiente a los requisitos *Ver Ocupación Hotel* y *Realizar Nueva Reserva*

En esta figura vemos cómo la información contenida en el listado de ocupación del hotel está repartida en tres clases navegacionales: *Cliente*, *Reserva* y *Habitación*. La agrupación de esta información en una única estructura lógica (página) se realiza mediante el carácter *en origen* de los enlaces entre ellas (ver la cabeza hueca de las flechas en la Fig. 5.10). Además, el hecho de activar los filtros estructurales *Cliente2Reserva* y *Reserva2Habitación* garantiza que esta información esté relacionada de la manera correcta. El último paso para modelar este listado es asociar un patrón de

navegación *Mostrar Todo* que presente todas las ocupaciones en la misma página.

Supongamos ahora que el diseñador decide refinar el modo de realizar una nueva reserva, y planea mostrar un formulario de búsqueda de habitaciones de un tipo determinado libres en un determinado período de tiempo. Una vez obtenido el listado de habitaciones libres, la aplicación debe darnos la opción de realizar la nueva reserva sobre cualquiera de ellas. Este nuevo modelo de interacción usuario-sistema, distinto al planteado en la Fig. 5.10, se plasma en OO-H en un nuevo diagrama de navegación de nivel 2, tal y como puede verse en la Fig. 5.11.

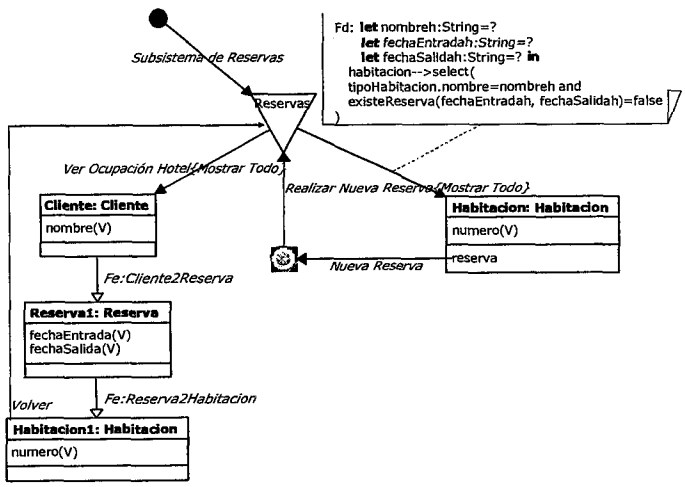

Figura 5.11. DAN de Segundo Nivel refinado correspondiente a los requisitos *Ver Ocupación Hotel* y *Realizar Nueva Reserva*

En esta figura vemos cómo en este refinamiento el diseñador ha incluido un menú que responde a los dos requisitos funcionales cubiertos en el DN de *Reservas*: *Ver Ocupación Hotel* y *Realizar Nueva Reserva*. La forma de modelar el listado de ocupación del hotel es idéntica a la mostrada en la Fig. 5.10. Sin embargo, en esta ocasión la realización de la reserva sigue un procedimiento distinto. En primer lugar un Filtro en Destino (Fd) selecciona las

habitaciones que, siendo de un determinado tipo, están disponibles en una determinada fecha. Tanto el tipo como las fechas son parámetros que el usuario debe introducir en tiempo de ejecución, aspecto que se modela mediante el símbolo *?*. También destaca el uso de la operación *existeReserva* (de tipo *isQuery=true*) como parte del filtro (fórmula OCL) que permite comprobar si una habitación está ocupada en un período comprendido entre dos fechas. El resultado de aplicar este filtro es una selección de las habitaciones que están libres. Una vez visualizado el listado de números correspondientes a dichas habitaciones disponibles, la aplicación proporciona acceso a la interfaz del servicio *reserva*, cuyo proceso de modelado será visto en el capítulo 6. Tras la recuperación del control por parte de la interfaz, ésta muestra de nuevo el menú con las dos opciones disponibles.

Por último, nos gustaría destacar cómo el cambio de modelo de interacción usuario-sistema puede producirse no sólo por refinamientos de uso sino por exigencias del dispositivo de acceso. En este caso, nuevamente habría que adaptar este modelo de navegación de diseño para que reflejase el nuevo modelo de interacción.

5.4 Influencia de los Enlaces Semánticos en el DAN

Tal y como hemos visto en la sección 5.3.7, el modelo de análisis de navegación (ver Fig. 5.1) determina la estructuración de primer nivel del Diagrama de Acceso Navegacional durante la actividad de diseño. En él (ver Fig. 5.9) vemos cómo las distintas unidades semánticas se materializan en Destinos Navegacionales que engloban a los constructores que dan respuesta a cada subgrupo de requisitos. Si explosionamos cada Destino Navegacional, su contenido incluye colecciones, clases y enlaces. Entre estos últimos, hay algunos que son una materialización directa de los enlaces semánticos del modelo de análisis. Si observamos por ejemplo la Fig. 5.10, vemos cómo los dos requisitos cubiertos aparecen como etiquetas de dos de los enlaces de diseño modelados.

De hecho la diferenciación entre enlaces semánticos y enlaces de diseño introduce una jerarquía en la importancia de los enlaces que conforman aplicación. De este modo, la existencia y homogeneidad[5] en la materialización de los enlaces semánticos debe ser preservada entre dispositivos de acceso, modelos de interacción y plataformas objetivo. Por otro lado los enlaces de diseño que no son fruto de esta materialización sí pueden verse afectados por éstos y otros factores, y por tanto pueden ser redefinidos para cada instanciación (implementación final) de la especificación de la aplicación.

En OO-H la homogeneidad de los enlaces semánticos se materializan en la preservación de dos rasgos principales. En primer lugar, la materialización de un enlace semántico se incluye de manera automática en menús de acceso directo del más alto nivel, proporcionando así un modo de entrada inmediato a los requisitos funcionales cubiertos por la aplicación. Además, la materialización de un enlace semántico induce un paso de navegación, es decir, tiene una *ubicación del efecto = destino* y un *modo de activación = manual*. Dicho de otro modo, este tipo de enlace asegura que las respuestas a dos requisitos funcionales distintos no aparecen en la misma página abstracta[6].

Por otro lado, OO-H define una regla de diseño que especifica que el tipo y cantidad de información contextual transmitida a través de un enlace de diseño generado a partir de un enlace semántico debe ser mínima. Puesto que los enlaces semánticos implican un cambio en la intención del usuario, no se deberían hacer demasiadas asunciones basándose en la intención que éste tenía hasta el momento. Además, a nivel de diseño de presentación (actividad que será introducida en el capítulo 8), la caracterización de un enlace de diseño como materialización de un enlace semántico puede inducir el uso de colores, fuentes, etc. más llamativos que permitan diferenciarlos.

[5] con homogeneidad nos referimos a la preservación de ciertas características y comportamiento asociado entre los distintos modelos de diseño de navegación. El número y valores de estas características depende de la propuesta metodológica que estemos aplicando.

[6] El concepto de página abstracta será tratado en profundidad en el capítulo 8.

Si atendemos al tipo de enlace, en OO-H los R-Enlaces suelen ser materializaciones de Enlaces Semánticos. Del mismo modo, los T-Enlaces que conectan constructores pertenecientes a Destinos Navegacionales distintos también suelen ser materializaciones de enlaces semánticos. Por último, los S-Enlaces son materializaciones de enlaces semánticos cuando (1) la cobertura de un requisito de usuario se produce por la mera invocación del método subyacente o (2) la vista por donde debe seguir navegando el usuario una vez ejecutado el servicio y visualizados los resultados supone un salto navegacional a un Destino Navegacional distinto.

Por último, nos gustaría destacar cómo esta categorización de enlaces es sólo un primer paso en el descubrimiento de nuevas relaciones semánticas (Bieber, 2000) que, capturadas mediante nuevos metadatos, puede incrementar de manera significativa el éxito de implantación de las aplicaciones hipermediales. Como ejemplo, y pensando en aplicaciones de comercio electrónico (donde uno de los objetivos operacionales es incrementar las ventas), la categorización de un enlace como un *proveedor de beneficio monetario* para la empresa puede causar cambios en la interfaz (orden de menús, disponibilidad del enlace, etc.) que apoyen su propósito de venta. Este tipo de categorización no tiene nada que ver, como podemos observar, con cómo ni a través de qué estructuras de información tiene que navegar el usuario con el fin de realizar una compra o, dicho de otro modo, es independiente del diseño de navegación.

5.5 Conclusiones del Capítulo

Este capítulo ha presentado las actividades de análisis y diseño de navegación, que producen como resultado el modelo de navegación de OO-H.

A nivel de análisis, hemos definido el concepto de *Navegación Semántica*, y hemos presentado los dos constructores que conforman el *Diagrama de Navegación Semántica* correspondiente: los enlaces semánticos y las unidades semánticas. Una unidad

semántica es una super-estructura que incluye la información, la funcionalidad y las estructuras navegacionales necesarias para cubrir un subconjunto de requisitos funcionales. Los enlaces semánticos por su parte capturan una posibilidad de cambio en la intención del usuario mientras navega por la aplicación. El diagrama resultante es independiente de plataforma, dispositivo y modelo de interacción usuario-sistema, hecho que permite la reflexión y el reuso de estructuras de navegación globales entre diferentes implementaciones de la misma aplicación.

A continuación, y como parte de la actividad de diseño de navegación, hemos definido el concepto de *Navegación Estructural*, y hemos presentado los constructores (clases, enlaces, colecciones y destinos navegacionales) y el método que permiten la construcción de un *Diagrama de Acceso Navegacional*. De entre dichos constructores, destaca la riqueza semántica de los enlaces navegacionales. Estos enlaces tienen asociados un conjunto de atributos: ámbito de aplicación, modo de activación y ubicación del efecto. Además, la navegación estructural que modelan puede ser enriquecida mediante patrones y filtros navegacionales.

Una de las características más importantes del modelo navegacional es que, tal y como veremos en el capítulo 9, contiene la información mínima necesaria para generar una interfaz operativa. Esta generación se consigue asumiendo valores por defecto para el resto de fases del método. Este rasgo del DAN permite al diseñador acortar el tiempo necesario para desarrollar prototipos de aplicaciones.

6. Modelado de Interfaces de Servicio en OO-H

No es la especie más fuerte la que sobrevive, ni la más inteligente, sino la más adaptable al cambio.

CHARLES DARWING

Como ya comentamos en el capítulo 1, una de las principales características de OO-H es la adopción de una perspectiva navegacional para abordar el modelado de aspectos referidos tanto a la introducción de parámetros como a la recuperación (y posterior visualización) de resultados en aplicaciones donde el tipo de interacción usuario-sistema requiere algo más que una mera activación de enlaces entre páginas. Este capítulo describe en detalle estas características y completa por tanto la visión dada en el capítulo 5 acerca de los conceptos relativos al diseño de la navegación de usuario a través del espacio de información.

Para ello, en la sección 6.1 comenzaremos clarificando el significado del concepto de *servicio* en el contexto de este trabajo. Este concepto puede ser clasificado siguiendo distintos criterios, tal y como veremos en la sección 6.2. En la sección 6.3 veremos el tipo de interfaz que OO-H es capaz de proporcionar a dichos servicios, y que puede ser de dos clases: interfaces simples e interfaces multipaso. Para el modelado de estas interfaces no triviales OO-H proporciona un constructor, el *Enlace de Servicio*, que, a nivel de diseño navegacional, incluye la semántica y notación que se detalla en la sección 6.3.1. Este constructor, como ya dijimos en el capítulo 5, permite tanto el modelado de la forma en que el usuario introduce valores para los parámetros como el modo en que la interfaz vuelve a recuperar el control de la aplicación, muestra el resultado de la invocación y permite reanudar la navegación del usuario a partir de esta visualización de resultados. En la sección

6.3.2 ilustraremos todos estos conceptos mediante su aplicación a nuestro Sistema de Gestión de Hoteles. Por último, la sección 6.4 presenta las conclusiones del capítulo.

6.1 El concepto de servicio en las aplicaciones web

Como en otros campos emergentes, la terminología asociada a la Ingeniería Web aún no está bien definida, y aún menos si nos restringimos a aspectos relativos a los modos de soportar la funcionalidad de las aplicaciones en web. En concreto el abuso que se ha producido en el uso de términos como *servicio* o, en su versión más reciente, *servicio web* para hacer referencia a aspectos que están relacionados pero que no son equivalentes ha creado una gran confusión.

El significado de estos términos varía, en primer lugar, según sean utilizados desde un punto de vista de negocio o tecnológico. Desde el punto de vista de negocio (Kirda *et al.*, 2001), un servicio web es una oferta empresarial global ofertada a través de internet, como puede ser una aplicación que posibilite la realización de reservas en línea. Por el contrario, desde un punto de vista tecnológico (Goeschka & Schranz, 2001) un servicio es fundamentalmente una interfaz que permite la invocación de un método en tiempo de ejecución. Esta interfaz proporciona acceso a porciones de código que pueden ser accedidas desde internet (algunas previo pago), y donde es el cliente el que proporciona la capa de presentación asociada a la interfaz de servicio. De manera aún más restringida, desde el punto de vista del Paradigma Orientado a Objetos, un servicio es aquello que ofrece una clase mediante cada una de sus operaciones (UML 1.4, 2001). En cualquier caso, un servicio desde el punto de vista de negocio puede involucrar más de un servicio tecnológico e incluso un sistema de información completo (Herzum, 2001).

En OO-H, y siguiendo la filosofía tecnológica que guía toda la aproximación, al hablar de *servicios* nos referimos a los servicios

tecnológicos, i.e. a interfaces que proporcionan acceso a módulos de funcionalidad o bases de datos[1] que coexisten bajo la capa de interfaz de una aplicación web para cubrir los requisitos de usuario[2]. En cuanto al término *Servicio Web*, en este trabajo lo reservaremos para referenciar una plataforma tecnológica concreta, que hace uso de un lenguaje y un conjunto de protocolos de acceso determinados.

6.2 Criterios de Categorización de Servicios en OO-H

OO-H identifica distintos criterios ortogonales que pueden inducir una clasificación de servicios. Entre ellos destacamos:

- Sincronía: da lugar a una diferenciación entre servicios síncronos y asíncronos.
- Agente Activador: genera una clasificación en servicios activables por la aplicación, por el usuario o por el entorno.
- Transaccionalidad: induce una separación entre servicios transaccionales y no transaccionales.
- Atomicidad: permite la diferenciación entre servicios atómicos y compuestos.

6.2.1 Sincronía

Los servicios síncronos son aquéllos en los que, una vez invocado el servicio, la aplicación espera hasta que se devuelve un resultado. Por el contrario, los servicios asíncronos son aquéllos en los que la aplicación no espera a recibir resultados, sino que sigue con

[1] Ejemplos de bases de datos que pueden ser ofrecidas como servicios incluyen DB2/XML Extender sobre DB2 (IBM) o los Servicios Dinámicos de Oracle.

[2] A lo largo de este trabajo utilizaremos el término *servicio* u *operación* como sinónimos que hacen referencia a estas interfaces, y reservaremos el uso de la palabra *método* para referirnos a la implementación de dicha interfaz.

su flujo normal de ejecución. El tratamiento de estos servicios requiere que la aplicación se convierta en observadora del entorno, para poder detectar cuándo se produce la respuesta del servicio invocado y poder ofrecérsela al usuario. Además, y con el fin de mejorar la facilidad de uso, esta oferta debería ser no intrusiva, es decir, informar al usuario de que la respuesta está disponible pero dejando que éste decida cuándo quiere visualizar el resultado de la invocación asíncrona. Por el momento OO-H se centra en el modelado de servicios síncronos, y deja la consideración de servicios asíncronos como trabajo futuro.

6.2.2 Agente activador

Por otro lado, los servicios pueden ser activables por la aplicación, activables por el entorno o activables por el usuario. Los servicios que sólo son invocados desde otros servicios (normalmente por motivos de modularidad, aumento de cohesión y/o disminución de acoplamiento) quedan fuera del ámbito de estudio de OO-H, ya que su gestión se encapsula dentro de la propia lógica de negocio de la aplicación. La activación de servicios por cambios en el entorno supone que la aplicación es capaz de desencadenar su ejecución, que puede suponer o no la introducción de parámetros por parte del usuario, sin que éste lo indique de manera expresa. Este tipo de activación será vista en el capítulo 7, cuando hablemos de la proactividad en la personalización. Por último, un servicio es activable por el usuario cuando éste tiene control explícito sobre su momento de activación. Son precisamente estos últimos servicios los que entran dentro del ámbito de responsabilidad de la actividad de diseño navegacional de OO-H.

6.2.3 Transaccionalidad

Otro aspecto relevante desde el punto de vista de OO-H es su carácter transaccional o no. OO-H distingue entre dos tipos de transacciones:

- B-transacciones: transacciones de la lógica de negocio. Suelen diseñarse para trabajar con múltiples usuarios que invocan servicios sobre un SGBD particular o una federación de SGBD. Además, normalmente son ejecutadas por un manejador de transacciones, que se ocupa de que el estado del sistema nunca sea inconsistente, y sus características son la Atomicidad, Consistencia, Aislamiento y Durabilidad[3].
- I-transacciones (Billard, 1998): son acciones atómicas asociadas a usuarios (no a sistemas) que pueden ejecutarse sobre múltiples sistemas que no tienen por qué conocerse entre ellos. A las características ACID de las transacciones tradicionales, las I-transacciones añaden dos más: la Privacidad y la Lealtad, convirtiéndose así en transacciones PLACID. Es importante destacar que una I-transacción representa una arquitectura de servicios federados, donde los posibles manejadores de transacciones de los lugares web involucrados no se conocen. Es por ello que en la actualidad se están realizando numerosos esfuerzos orientados hacia la estandarización de protocolos de transacción que permitan que un gestor de transacciones global dirija el proceso de invocación y vuelta atrás de servidores locales arbitrarios (XAML, 2001; XLANG, 2001).

En OO-H las B-transacciones resultan transparentes, ya que son responsabilidad de los módulos de lógica subyacentes. Sin embargo, el concepto de I-transacción sí resulta relevante, ya que su carácter integrador de servicios independientes, y por tanto no diseñados para actuar en conjunto, hace necesaria la provisión de mecanismos adicionales de coordinación y control transaccional al nivel en que se pretende realizar su integración.

6.2.4 Atomicidad

Por último, un servicio es atómico cuando proporciona una interfaz que puede ser capturada en el modelo de dominio. Por el

[3] Propiedades ACID: Atomicy, Consistency, Isolation, Durability

contrario, los servicios compuestos son aquéllos que integran varios servicios atómicos, cada uno de los cuales, aun pudiendo ser invocado de manera independiente, debe colaborar con el resto para satisfacer la necesidad de algún usuario en algún momento de la vida de una aplicación. Además, los átomos de un servicio compuesto pueden ser ofertados todos ellos por un mismo proveedor (lo que facilita su federación) o, por el contrario, pertenecer a distintos proveedores.

Los servicios compuestos presentan toda una serie de nuevos retos, entre los que, a nivel técnico, destacamos su federación, la gestión (si es necesario) de su carácter transaccional (conformando las I-transacciones que veíamos en la sección 6.2.3), la gestión de errores o, bajando el nivel de abstracción, el manejo de las dependencias funcionales que pueden existir entre sus átomos.

Algunas aproximaciones recientes (Rodríguez & Díaz, 2002) han profundizado en estos aspectos y han desarrollado modelos para definir el flujo de ejecución y las dependencias existentes entre los servicios atómicos que conforman el servicio compuesto. Todos estos temas quedan fuera del ámbito del presente trabajo.

6.3 Las Interfaces de Servicio en OO-H

Tal y como comentábamos en el capítulo 2, en los últimos tiempos numerosas propuestas hipermediales han incluido las operaciones CRUD como constructores básicos de modelado (Mecca *et al.*, 1999a; Schwabe & Almeida, 1999; Ceri *et al.*, 2000). Sin embargo ninguna propuesta, hasta lo que alcanza nuestro conocimiento, proporciona mecanismos específicos para modelar interfaces de operaciones de propósito general, del tipo de las demandadas en las aplicaciones actuales, y donde el tipo de interacción es mucho más complejo que la mera cumplimentación de un conjunto de campos de texto.

OO-H palía este problema al considerar a los servicios como ciudadanos de primera categoría (Cachero *et al.*, 2000a; Cachero *et al.*,

2001b; Cachero & Gómez, 2002), y encapsular las características particulares de su interfaz en un constructor que permite especificar interfaces complejas mediante el uso de técnicas de navegación similares a las que se utilizan para visualizar y/o interactuar con el resto de información del sistema.

Esta concepción navegacional de la invocación de servicios es compartida por autores como (Bieber *et al.*, 1999). En OO-H, este hecho proporciona como ventaja fundamental la posibilidad de definir distintas interfaces de usuario para los distintos actores del sistema. En efecto, en numerosas ocasiones el modo de interacción para un parámetro determinado varía en función del tipo de usuario que esté utilizando la aplicación. Pensemos por ejemplo, en nuestro Sistema de Gestión de Hoteles, en la invocación del servicio *Realizar Nueva Reserva*, y supongamos que puede ser invocado por el cliente o por el recepcionista del hotel. En el primer caso parece lógico pensar que el parámetro *cliente* asociado al servicio adopte el valor del cliente que está interactuando con la aplicación. En el segundo caso por el contrario parece lógico pensar que el recepcionista debería tener un mecanismo para poder seleccionar el cliente a nombre del cual se debe hacer la reserva.

Para el modelado de interfaces de servicio, tal y como vimos en el capítulo 4, el primer paso consiste en la captura, durante la actividad de diseño dominio, de un conjunto de parámetros referentes a la interfaz de la operación que se desea invocar, y que incluye el nombre del método subyacente, sus parámetros de entrada y salida sus correspondientes tipos. Además, esta especificación puede enriquecerse con precondiciones y postcondiciones, que reflejan restricciones tanto sobre la posibilidad de invocación del servicio como sobre el modo de determinar el éxito/fracaso de la invocación.

A partir de este modelo de dominio, el objetivo de la actividad de diseño navegacional por lo que respecta a los servicios es el enriquecimiento de esta especificación con información referente a cómo el usuario debe interactuar con el servicio (i.e. cómo debe introducir los parámetros y cómo puede visualizar los resultados de la invocación).

A continuación veremos cómo OO-H modela éstas y otras situaciones.

6.3.1 El Enlace de Servicio

El modelado de servicios en OO-H se realiza mediante el S-Enlace que, tal y como vimos en el capítulo 5, define el *conjunto* de caminos de navegación que permiten la interacción con módulos de lógica preexistentes.

Los enlaces de servicio son enlaces definidos sobre operaciones pertenecientes a una clase navegacional. Su origen es por tanto una operación, y su destino puede ser un destino navegacional, una clase navegacional o una colección. La representación gráfica de este constructor fue presentada en la Fig. 5.2.

Los principales rasgos capturados en el enlace de servicio de OO-H incluyen:

- Introducción de valores de entrada asociados a cada uno de los parámetros.
- Invocación del método.
- Recuperación del control y visualización de resultados o notificación de errores.
- Especificación del punto de partida de la navegación una vez se ha ejecutado el servicio y se han presentado los resultados.

En primer lugar, cada parámetro de entrada (i.e., caracterizado como de tipo *in* durante la actividad de diseño de dominio) puede tener asociado uno o más caminos de navegación que permiten seleccionar los valores de entrada al servicio.

El segundo paso consiste en la invocación del método propiamente dicha. Esta invocación es responsabilidad del compilador de modelos que acompaña a OO-H, y por tanto será comentada en el capítulo 9.

Una vez ejecutado el método subyacente, el S-Enlace puede incluir nuevamente uno o más caminos que, asociados a los parámetros de tipo *out* definen la manera en que se debe mostrar el resultado de la ejecución del servicio.

Por último, el enlace de servicio debe dirigir al usuario hasta un punto en que pueda continuar su interacción con la aplicación.

La introducción de parámetros. Los S-Enlaces comparten con el resto de enlaces un conjunto de atributos comunes, que fueron comentados en la sección 5.3.5 y que incluyen un *modo de activación*, una *ubicación del efecto* y un *ámbito de aplicación*.

Además, los S-Enlaces tienen otro conjunto de características propias, entre las que destacamos los **Modos de Introducción** que, asociados a los distintos parámetros de cada operación, permiten modelar la entrada de valores por parte del usuario, y que dan lugar a:

- *Parámetros Ocultos.* Estos parámetros no aparecen en la interfaz, y deben tomar por tanto el valor por defecto (si se proporciona, bien sea a nivel de diagrama de clases o a nivel de DAN) o un valor nulo en otro caso.
- *Parámetros constantes.* Su única diferencia respecto a los parámetros ocultos es que se considera que su valor es relevante para la tarea que esté realizando el usuario, y por tanto se presentan en la interfaz, siempre en modo no editable.
- *Parámetros de introducción inmediata.* Son parámetros para los que no se proporciona ningún tipo de guía que oriente al usuario en cuanto a los valores que puede adoptar, y que a nivel de presentación son soportados por campos de introducción libre de texto (o mecanismos similares)[4].
- *Parámetros de Introducción por Selección.* Son parámetros para los que el usuario introduce un valor de entre un conjunto de

[4] Normalmente este tipo de parámetros requiere la aplicación, antes de proceder a la ejecución del método, de las comprobaciones necesarias para asegurar que el valor introducido es válido.

valores predefinidos. Si estos valores se determinan en tiempo de diseño, hablamos de una *selección no contextual*, mientras que si se extraen en tiempo de ejecución del estado en que se encuentre el sistema en ese momento hablamos una *selección contextual*. Los parámetros que pertenecen a un tipo estereotipado como <<enumeration>> en el diagrama de clases son un candidato claro a este modo de introducción.

- *Parámetros de introducción por navegación de usuario.* Este modo de introducción es sin duda el más interesante desde el punto de vista de OO-H. El modelado de distintos tipos de aplicaciones nos ha hecho detectar que, en multitud de ocasiones, la introducción de un valor en un método supone una reutilización o incluso una definición de nuevos caminos de navegación en la aplicación. Basta pensar en la introducción de una reserva en nuestro sistema de gestión de hoteles: la elección de un cliente y una habitación se puede hacer tediosa si no disponemos de mecanismos que nos estructuren y nos permitan filtrar toda la población de clientes y habitaciones de nuestro sistema.

En el caso de la introducción por navegación, la asociación de esos caminos de navegación a un determinado parámetro de entrada se puede producir de dos maneras:

- Definición del enlace inicial y final del camino de navegación (útil cuando deseamos reutilizar todo lo definido entre ese enlace inicial y final).
- Introducción, por medio de fórmulas OCL, de caminos de navegación específicos. Este mecanismo es útil cuando la porción a reutilizar contiene muchas alternativas de navegación, y estamos interesados sólo en una (o un pequeño subconjunto) de ellas.

Valores por defecto. Por otro lado, cada parámetro de tipo *in* puede tener asociado un valor por defecto, que puede provenir de dos fuentes: (1) del diseño de dominio (valores por defecto de los parámetros de los servicios en el diagrama de clases) o (2) de una expresión OCL asociada al parámetro, y que puede

ser contextual (expresión que referencia un valor del contexto de trabajo del usuario, y que por tanto debe ser evaluada en tiempo de ejecución) o no contextual (expresión constante).

Si se especifica un valor por defecto a nivel de DAN, las reglas de precedencia de OO-H determinan que es el último valor especificado (en este caso el de nivel DAN) el que sobreescribe al anterior (el proporcionado en el diagrama de clases de dominio).

Interfaces Multipaso. Por último, la introducción de parámetros puede producirse mediante una estructura de interfaz de invocación de servicio centralizada o distribuida. Una estructura centralizada, típica de interfaces de servicio sencillas, supone la existencia de una página núcleo donde se van recogiendo los valores de los distintos parámetros, independientemente de su modo de introducción. Este tipo de interfaces son denominadas en OO-H *Interfaces de Operación Simple*. Por el contrario, si nos encontramos una estructura de invocación en la que subconjuntos de valores se van recogiendo en distintas páginas (normalmente relacionadas de una manera lineal) y donde el usuario puede ir adelante y atrás entre ellas tantas veces como considere necesario[5] estaremos ante lo que OO-H denomina *Interfaces de Operación Multipaso*. Con el fin de soportar ambos tipos de interfaz, OO-H asocia a sus enlaces de servicio un valor etiquetado *multipaso=no/sí* que determina si estamos interesados en modelar o no una interfaz de estas características. En el caso de encontrarnos ante interfaces multipaso, la activación de este valor etiquetado permite asociar un ordinal a cada parámetro que indica el paso (dentro de esa estructura lineal) en el que se va a solicitar la entrada de valor correspondiente. De este modo es posible agrupar los parámetros de manera arbitraria.

[5] Este tipo de estructura es muy común en el diseño de la invocación de operaciones de compra en aplicaciones de comercio electrónico. En estas interfaces, en cada paso de lo que comúnmente se denomina un *proceso de salida* se introduce y/o valida un subconjunto de la información necesaria para efectuar la compra. Es importante hacer notar que esta compra representa una sola operación desde el punto de vista del usuario, independientemente de cómo está implementada (con uno o varios métodos, con control transaccional, etc. en el módulo de lógica subyacente.

Recuperación del control y visualización de resultados o notificación de errores. Una vez completado el servicio, la visualización de los valores correspondientes a los parámetros de retorno (caracterizados como de tipo *out* a nivel de modelo de dominio) sigue la misma filosofía que la de los parámetros de entrada, aunque su especificación es algo más sencilla.

Los parámetros de salida sólo pueden ser de tres tipos: *ocultos*, *constantes* o *navegables*, y este último sólo si el parámetro devuelto es un objeto o un conjunto de ellos. La existencia de parámetros de retorno *constantes* o *navegables* determinan su visualización.

En caso de tratarse de parámetros navegables (objetos), el modo mostrar su estado (la información que contienen sus atributos) es mediante la asociación de una fórmula OCL que, en este caso, siempre supone la reutilización de un enlace del modelo navegacional cuyo destino es necesariamente una clase navegacional instanciada a partir de la clase de dominio del objeto devuelto. Por ejemplo, supongamos que hemos definido la interfaz del servicio *Cliente.crear(in nombre,in dirección, in teléfono, out nuevoCliente)*. Para poder visualizar el objeto cliente creado, el enlace de navegación asociado al parámetro de salida *nuevoCliente* debe estar definido en el DAN sobre una clase navegacional creada a partir de la clase de dominio *Cliente*, y serán las características de este enlace (ubicación del efecto, patrón de navegación) las que se apliquen para visualizar el objeto.

Continuación de la navegación por el sistema. Una vez que los resultados deseados han sido presentados, el usuario puede continuar la navegación a través del sistema por la clase navegacional a la que apunta el S-Enlace. Para ello, este constructor incluye otro conjunto de parámetros que determinan la ubicación del efecto y el parámetro de navegación aplicable a esta continuación de la navegación[6].

[6] De hecho, en versiones anteriores del método, el modo en que el usuario continuaba la navegación se establecía mediante un nuevo tipo de enlace, el *Enlace de Respuesta*. El motivo de haber integrado ambos constructores es que todo enlace de servicio tiene obligatoriamente asociado un enlace de respuesta, por lo que hemos considerado que su fundición resulta más intuitiva para el diseñador.

6.3.2 Aplicación al Caso de Estudio

Ya hemos comentado cómo la configuración de interfaz de servicio más sencilla se basa en un solo paso de interacción, que se materializa en una sola pantalla de introducción de datos. Con el fin de ilustrar este tipo de interfaz, supongamos que hemos modelado un nuevo listado de reservas (compuesto por la fecha de entrada, fecha de salida, cliente, habitación) mediante el DAN presentado en la Fig. 6.1, que da lugar a una página de características similares a la presentada en la Fig. 6.5.

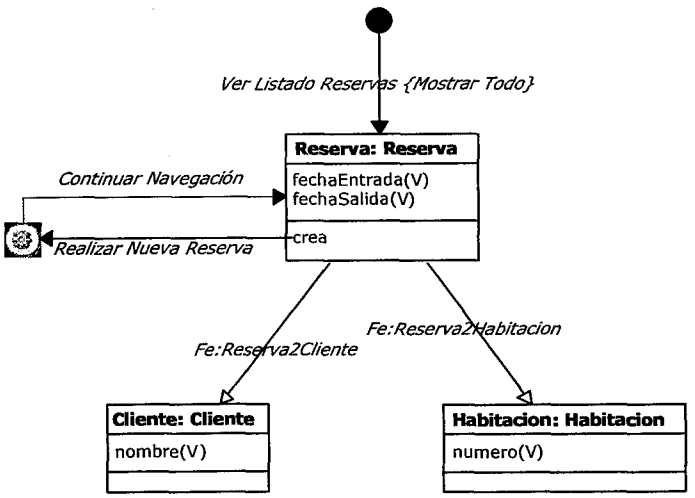

Figura 6.1. DAN correspondiente a una posible respuesta a los requisitos *Ver Ocupación Hotel* y *Realizar Nueva Reserva*

En este diagrama, observamos cómo el S-Enlace *Realizar Nueva Reserva* está asociado al servicio *Reserva.crea(in fEntrada:Date,in fSalida:Date,in cliente:Cliente,in habitacion:Habitacion,out correcto:Boolean)*, cuya interfaz fue definida durante la actividad de diseño de dominio. De la definición de esta interfaz de dominio podemos colegir que el alta por parte del recepcionista de una reserva requiere la introducción explícita de cuatro parámetros: *fEntrada* (de tipo fecha), *fSalida* (de tipo fecha), *cliente* (de ti-

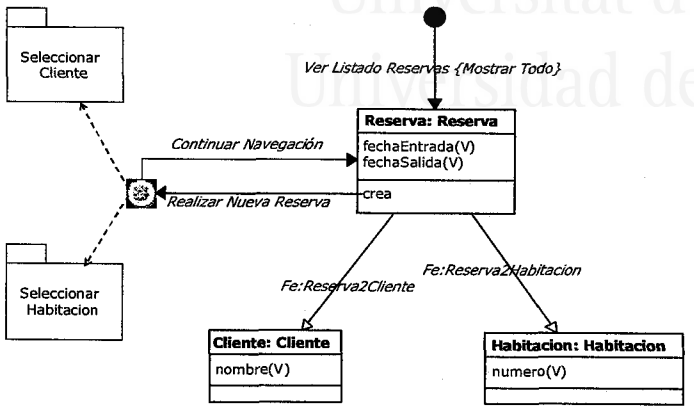

Figura 6.2. DAN refinado para incluir los caminos de introducción de parámetros por navegación

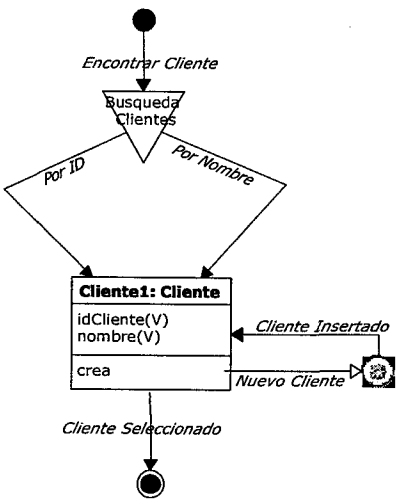

Figura 6.3. Explosión del DN *Seleccionar Cliente*

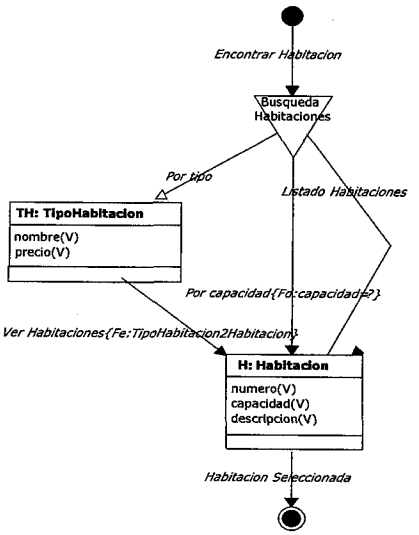

Figura 6.4. Explosión del DN *Seleccionar Habitacion*

po Cliente) y *habitacion* (de tipo Habitación), así como que el método subyacente devuelve un parámetro *correcto* (de tipo booleano) que adquiere el valor de *cierto* si la reserva se ha realizado con éxito, y *falso* en otro caso.

A la hora de modelar la interfaz, supongamos que deseamos obtener una estructura como la presentada en las Figs. 6.5 a 6.12.

Para ello tenemos que realizar los siguientes pasos:

- Introducir un enlace de servicio asociado a la operación *Reserva.crea*.
- Establecer algunas de sus características genéricas: *ámbito de aplicación=universal*[7], *modo de activación=manual* (ya que deseamos que el usuario indique explícitamente que desea introducir una reserva) y *ubicación del efecto=destino* (lo que hará que se visualice el formulario de entrada de parámetros en una nueva página). El resultado de esta configuración es la inclusión en la página que contiene el listado de reservas (ver Fig. 6.5) de un botón o mecanismo similar para permitir la invocación del servicio de crear nueva reserva.
- Modelar la introducción de parámetros
  - Establecer un *Modo de Introducción = inmediato* para los parámetros *fEntrada* y la *fSalida*.
  - Establecer un *Modo de Introducción = navegacion* para los parámetros *cliente* y *habitacion*. El resultado de estos modos de introducción es un formulario como el que puede ser visto en la Fig. 6.6.
  - Diseñar o reutilizar los caminos de navegación necesarios para seleccionar el cliente y la habitación deseados. En nuestro ejemplo, no existe ningún camino navegacional que pueda ser utilizado para este propósito, por lo que debemos modelarlos. Para ello, en la Fig. 6.2 se han introducido dos nuevos DN:

[7] Éste es el valor predefinido en OO-H para los *métodos de clase* (i.e. aquéllos que no se aplican sobre instancias concretas del sistema), tal y como se definen en UML. Para los métodos de instancia por el contrario el valor predefinido es *ámbito de aplicación=simple*.

Figura 6.5. Boceto de página correspondiente al listado de las reservas del Sistema de Gestión de Hoteles

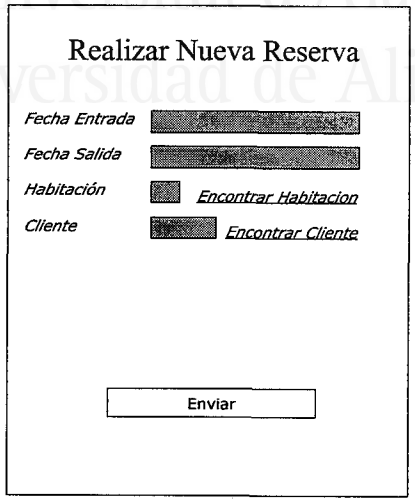

Figura 6.6. Boceto de página correspondiente al formulario de entrada de parámetros del servicio

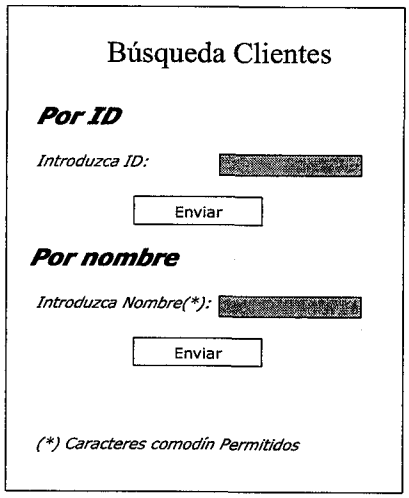

Figura 6.7. Boceto de página correspondiente a los formularios de búsqueda de clientes

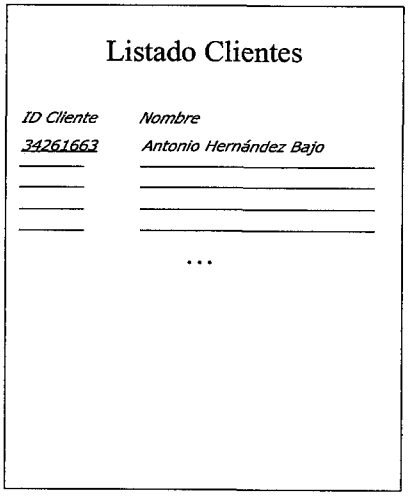

Figura 6.8. Boceto de página correspondiente al listado de clientes que cumplen criterio de búsqueda

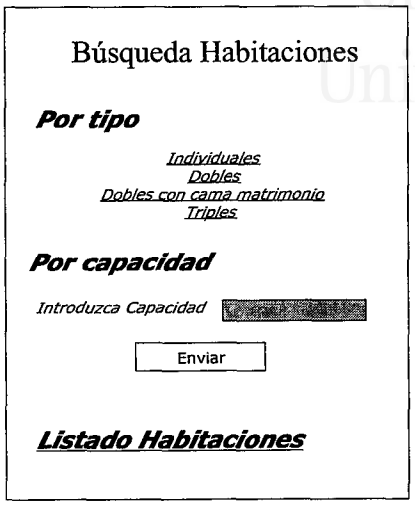

Figura 6.9. Boceto de página correspondiente a los formularios de búsqueda de habitación

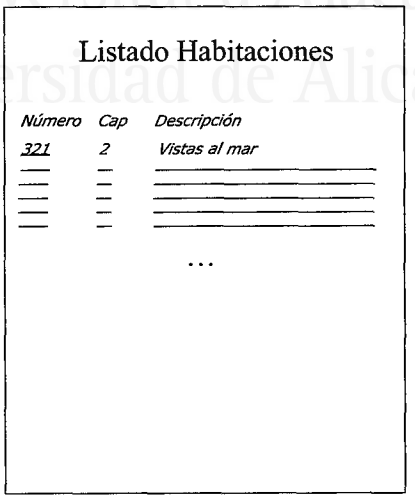

Figura 6.10. Boceto de página correspondiente al listado de habitaciones que cumplen criterio de búsqueda

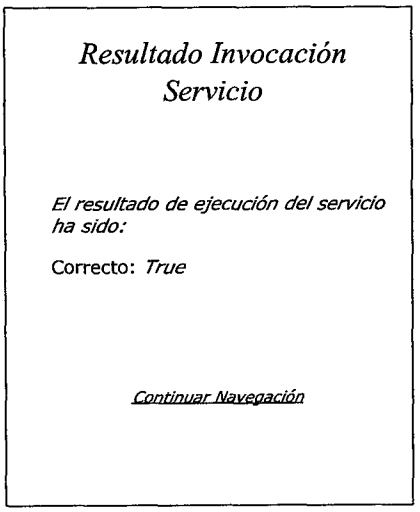

Figura 6.11. Boceto de página correspondiente a la visualización de resultados de la invocación del servicio

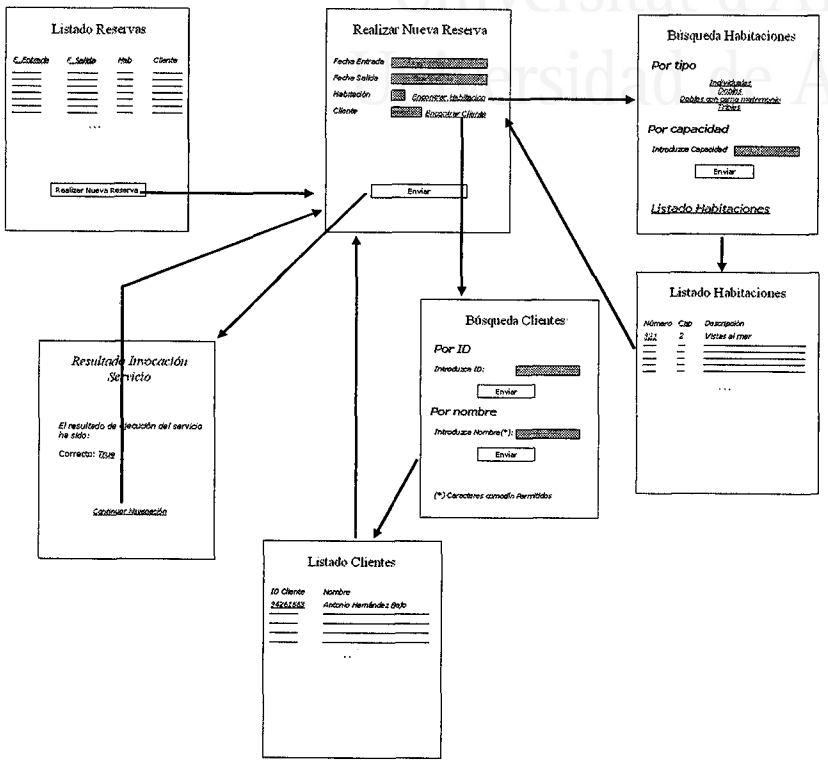

Figura 6.12. Estructura de interconexión entre los bocetos de página

Seleccionar Cliente y *Seleccionar Habitacion*. Las flechas punteadas entre el icono del servicio y estos DN indican que el modelado de la introducción de parámetros del servicio depende del contenido de dichos DN[8]. Los diagramas correspondientes a la explosión de esos dos destinos navegacionales pueden ser vistos en las Figs. 6.3 y 6.4.

Con el fin de recordar al lector la semántica de los constructores NAD, la Fig. 6.4 muestra cómo el cliente entra en el DN *Seleccionar Habitacion* mediante el R-Enlace *Encontrar*

[8] La introducción de las dependencias de navegación se realiza de manera automática en OO-H en caso de ser necesario cuando se asocia un enlace a un parámetro cuyo modo de introducción es por navegación

Habitacion. En ese punto, un menú *Busqueda Habitacion* proporciona acceso a tres posibilidades: buscar *por tipo* de habitación, buscar *por capacidad* o elegir la habitación a partir de un *listado habitaciones*. Tras elegir la habitación, el enlace *Habitacion Seleccionada* devuelve el control al nivel superior, es decir, a la pantalla de introducción de parámetros que veíamos en la Fig. 6.6, y la habitación elegida es asociada al correspondiente parámetro. La Fig. 6.3 permite, de forma análoga, la búsqueda de clientes, y tiene como peculiaridad la posibilidad que ofrece de que el recepcionista dé de alta un nuevo cliente si éste nunca antes se había alojado en nuestro hotel. En nuestro ejemplo, un esbozo de las páginas generadas por el DAN correspondiente al DN *Seleccionar Cliente* pueden ser vistas en las Figs. 6.7 y 6.8. Por su parte, las páginas correspondientes al DN *Seleccionar Habitacion* pueden ser vistas en las páginas 6.9 y 6.10.

- Asociar a los parámetros *cliente* y *habitacion* los enlaces de origen y destino correspondientes. De este modo la aplicación ofrecerá las posibilidades de navegación contenidas entre ese origen y ese destino para seleccionar el cliente y la habitación deseada. La asignación del enlace de entrada de cada DN al parámetro de servicio correspondiente causa el enlazado de páginas entre el formulario de entrada de parámetros (ver el enlace *Elegir Cliente* y *Elegir Habitacion* asociado al cliente y a la habitación en la Fig. 6.6) y estas nuevas páginas, tal y como podemos ver en la Fig. 6.12.

• Modelar la visualización de resultados

- Establecer el modo de visualización del parámetro de salida *correcto* como constante, para que la interfaz nos muestre su valor pero no nos permita modificarlo. El resultado de esta acción puede ser visto en la Fig. 6.11.

• Modelar la continuación de la navegación

- Establecer como destino del enlace de servicio la misma clase que el origen (*Reserva*, ver Fig. 6.2). Esto indica que, una vez realizada la reserva y visualizados el resultado de la invocación

del método, el usuario volverá al listado de reservas en que se encontraba cuando inició el método (que ahora incluirá la nueva reserva introducida), y desde ahí podrá continuar navegando por el sistema.

– Establecer el patrón de navegación de vuelta a la clase *Reserva* como *Mostrar Todo*. Esta configuración determina la inclusión de un botón o mecanismo similar en la página de resultados (ver Fig. 6.11) que nos lleva de vuelta al listado de reservas inicial.

6.4 Conclusiones del capítulo

Este capítulo completa el diagrama de acceso navegacional que fue introducido como parte del modelo navegacional en el capítulo 5. En él se ha presentado la aproximación navegacional adoptada en OO-H para el modelado de interfaces de servicio. Para ello, se ha comenzado presentando los distintos tipos de servicios, atendiendo a diversos criterios: sincronía, agente activador, transaccionalidad y atomicidad. A continuación, se ha presentado el enlace de servicio como principal constructor para el modelado de servicios síncronos, activables por el usuario, atómicos y no transaccionales, y se ha mostrado cómo con él es posible modelar la introducción de parámetros, la visualización de resultados y la continuación de la navegación a través de la aplicación. Además, se ha ilustrado cómo el modelado de estas interfaces se basa en la definición de interfaz proporcionada durante la actividad de diseño de dominio de la aplicación. Una taxonomía de tipos de introducción y la posibilidad de separación de parámetros en estructuras lineales que permitan agrupar parámetros relacionados en lo que OO-H denomina interfaces multipaso completa la propuesta. Esta aproximación incrementa el nivel de abstracción al que operan la mayoría de las aproximaciones de modelado web, y proporciona, mediante la asociación de caminos navegacionales preexistentes en la aplicación, un potente mecanismo de reuso de esta navegación.

7. Modelado de estrategias de personalización en OO-H

El efecto de la libertad en los individuos es que ahora pueden hacer lo que deseen - deberíamos identificar sus gustos, si es que buscamos reconocimiento

EDMUND BURKE. FILÓSOFO IRLANDÉS

Como ya comentamos en el capítulo 1, existen varios factores que influyen sobre todo el ciclo de desarrollo de la aplicación, de entre los cuales la *facilidad de uso*[1] juega un papel preponderante. En efecto, las características de la audiencia y la idiosincrasia de la web requieren la adopción de estrategias cada vez más agresivas para incrementar el grado de fidelidad de los usuarios, con el fin de garantizar el éxito de implantación de nuestra aplicación. Este éxito, en contra de lo que se podría pensar en un principio, va más allá de su excelencia tecnológica: ya no basta con que una aplicación cumpla con su cometido y lo haga de forma eficiente; la experiencia del usuario cuando está trabajando en ella tiene que ser también más positiva que la que tendría si accediese a otra aplicación sustitutiva que está a una sola pulsación de ratón.

En este capítulo comenzaremos por proporcionar (ver sección 7.1) una visión general de los distintos factores que intervienen en la definición de una política global de personalización. A continuación, en la sección 7.2 pasaremos a detallar el papel que juega esta política de personalización en el ámbito de los métodos hipermediales, así como los principales mecanismos que éstos integran para soportarla. Dentro de este conjunto, una visión general de los mecanismos ofrecidos por OO-H serán detallados en la sección 7.3. Estos mecanismos son tres: un modelo de usuario explícito

[1] *usability*

(ver sección 7.4), un modelo de referencia preparado para dar soporte a distintas técnicas de personalización (ver sección 7.5) y un proceso de diseño que tiene en cuenta los requisitos de personalización (ver sección 7.6) y que se apoya en estos modelos de usuario y de referencia. Por último, en la sección 7.7 presentamos las conclusiones del capítulo.

7.1 Contexto de la Personalización

Tradicionalmente el diseño de aplicaciones hipermediales ha sido un campo para informáticos e ingenieros en general. Esto ha causado que durante mucho tiempo se haya supuesto que una idea novedosa y una base tecnológica puntera eran en general suficientes para asegurar el éxito de una aplicación en internet. El fracaso de muchas empresas virtuales demuestra, desde nuestro punto de vista, lo equivocado de esta concepción. La percepción que el usuario tiene de una aplicación no se basa en la tecnología subyacente (que para él es algo transparente), e incluso la idea más novedosa puede verse eclipsada por un acceso ineficiente o una desorganización cognitiva a nivel de interfaz, que frustra e inhibe al usuario en su propósito de vuelta a nuestra aplicación.

En este sentido, existen numerosos factores que pueden influir en la facilidad de uso, como es la adición de motores de búsqueda, el uso de constructores navegacionales intuitivos, limitación de los niveles de anidación de la información, etc. Muchos de ellos tienen que ver con el uso correcto de constructores en la fase de diseño de presentación, y para su preservación existen numerosas guías y consejos de diseño (Nielsen *et al.*, 2001) que quedan fuera del ámbito del presente trabajo. Otras características sin embargo actúan a un nivel de abstracción mucho más elevado, llegando a influir en todo el ciclo de desarrollo de la aplicación, desde la recogida de requisitos a la propia arquitectura de ejecución. De entre ellas destacan la calidad navegacional (entendida como la uniformidad y coherencia en las métaforas de navegación) y, sobre todo, un conjunto de características relacionadas con el

grado de personalización de las aplicaciones. Este tipo de características presentan un interés especial desde el punto de vista de los métodos hipermediales, y su consideración explícita obliga a revisar las distintas actividades del método.

7.1.1 Disponibilidad de Fuentes de Información

La personalización de aplicaciones, entendida como la *capacidad de alteración dinámica con el fin de proporcionar al usuario la impresión de estar trabajando con una aplicación específicamente diseñada para dar satisfacción a sus necesidades particulares*[2], se puede producir en base a tres tipos principales de fuentes de información (Ceri *et al.*, 1999; Fraternali & Paolini, 2000; Kappel *et al.*, 2000):

- **Datos preexistentes**, normalmente importados de fuentes externas[3].
- El **perfil y las preferencias** de cada usuario individual. Esta información puede haber sido (Koch, 2001):
  - Introducida por el diseñador de la aplicación mediante el uso de técnicas de adquisición de tipo *estereotipado*.
  - Introducida por el propio usuario mediante el uso de técnicas de adquisición de tipo *entrevista*.
  - Inferida por la aplicación en base a la actividad del usuario en el sistema, mediante técnicas de adquisición de tipo *observación*.
- El **contexto** en el que se produce la interacción usuario-aplicación: localización, momento, características de la red, etc.

Por otro lado, la obtención de datos de cada una de estas fuentes se puede producir en base a distintas *técnicas de adquisición*.

[2] Este concepto es denominado por algunos autores *One to One Delivery* (Ceri *et al.*, 1999)

[3] El tratamiento de este tipo de fuentes queda fuera del ámbito de estudio del presente trabajo, y por tanto será obviado en el resto del capítulo.

Además, la activación del proceso de adquisición se puede producir ante un evento de usuario o bien en base a un chequeo, continuo o a intervalos regulares, de dichas fuentes. En el primer caso hablamos de *técnicas de adquisición reactivas*, mientras que en el segundo hablamos de *técnicas de adquisición proactivas*.

En cualquier caso, la información recogida durante este proceso de adquisición es la base sobre la que se definen los distintos tipos de técnicas de personalización, que presentamos a continuación.

7.1.2 Técnicas de Personalización

En la actualidad, el número y variantes de técnicas de personalización existentes parece incontable (ACM, 2000).

De entre ellas, algunas de las más utilizadas, y que pueden afectar a cualquier nivel de la interfaz (contenido, navegación y presentación), son (Pavón, 2001; Personalization Consortium, 2002):

- **Filtrado Simple**. El Filtrado Simple consiste en la definición de grupos de usuarios, a los que se asocian vistas particulares de la aplicación. Estos grupos de usuarios pueden estar predefinidos (e.g. en base a roles), o bien crearse en base a determinados atributos del usuario o del contexto (URL de conexión, género, edad, etc.). Como ejemplo, se puede diseñar una vista restringida de la aplicación destinada a niños menores de doce años. Otro ejemplo clásico es el análisis de la URL para determinar el idioma en que se debe presentar la aplicación.
- **Filtrado por contenido**. En este tipo de técnica, y partiendo de un conjunto de categorías, se determina la pertenencia o no de los objetos a cada una de dichas categorías, y se muestran aquéllos que entren en el grupo de categorías de interés de cada usuario. Esta técnica tiene como principal inconveniente el gran grado de objetividad que requiere. No obstante, se suele utilizar en determinados tipos de sistemas, como son los sistemas de clasificación de documentos.

- **Filtrado colaborativo**. Esta técnica, que permite aplicar criterios subjetivos y por tanto es muy utilizada como soporte de sistemas de recomendación, se basa en la actividad de otros usuarios de comportamiento similar al usuario actual para determinar aspectos como cuál se prevé que sea la próxima acción que el usuario va a realizar en el sistema o en qué productos puede estar interesado, proporcionando así una guía de navegación a nivel global o local en el sistema (Koch, 2001). El filtrado colaborativo está en el origen del concepto de la Navegación Social (Svensson *et al.*, 2001), que pretende simular los mecanismos de orientación de los que dispone el usuario en el mundo real. Esta técnica, que proporciona una capacidad de adaptación elevada a la aplicación, tiene como principales inconvenientes el alto coste computacional al que puede dar lugar en presencia de un alto volumen de usuarios y el bajo grado de fiabilidad de los algoritmos ante un volumen de información reducido, lo que obliga a establecer estrategias alternativas mientras no se disponga de dichos datos.
- **Perfilado** (*Profiling*). Por último, existen técnicas que, a partir de un identificador proporcionado por el propio usuario de manera implícita, permiten la personalización de la aplicación en base a la actividad del usuario en otras aplicaciones web. El uso de este identificador global limita los problemas asociados a la seguridad y privacidad de datos personales. Sus principales inconvenientes son, por un lado, la necesidad de que el propio usuario obtenga y proporcione este identificador global, y por otro la necesidad de intercambio e integración de esa información proveniente de fuentes externas para su posterior procesado.

Definición de Reglas de Asociación. Para la materialización de estas políticas, un posible mecanismo lo constituyen las *Reglas de Asociación*, entendidas como una definición de un conjunto de criterios que, cuando son cumplidos por un determinado usuario y (opcionalmente) ante la ocurrencia de un determinado evento, desencadenan una acción.

En el seno de OO-H (Cachero *et al.*, 2002c), hablamos de **Reglas Reactivas** cuando la acción de personalización es directamente causada por el propio usuario mientras está interactuando con la aplicación (e.g. activación de un enlace en la interfaz). Un ejemplo clásico de regla reactiva se da en los sistemas de comercio electrónico, donde a menudo se definen reglas de productos complementarios que son ofrecidos ante la compra del producto principal. En este caso, la compra de un producto (es la acción del usuario la que genera el evento desencadenante del chequeo de la regla) causa que la aplicación compruebe lo que han comprado otros usuarios que adquirieron ese producto (chequeo de la información adquirida por la aplicación mediante algún tipo de *técnica de adquisición*) para generar una lista personalizada (acción de personalización). Del mismo modo, hablamos de *Reglas Proactivas* cuando la acción de personalización se produce como consecuencia directa de una notificación por parte del entorno de interacción de que ha sufrido un cambio, que causa una reacción de la interfaz independientemente del usuario. Un ejemplo es un aumento en el ancho de banda de la conexión entre el usuario y la aplicación (no existe una acción de usuario que genere el evento desencadenante del chequeo de la regla), que puede permitir que la página del usuario se auto-refresque (acción de personalización) para darle acceso a contenidos (e.g. vídeos) que no se le ofrecían con el ancho de banda anterior.

Estas reglas pueden estar:

- Embebidas en el resto de la especificación del sistema.
- Externalizadas, normalmente mediante el uso de ficheros externos que pueden ser modificados sin tener que recompilar la aplicación.

La externalización tiene como principal ventaja la disminución del acoplamiento (con la consiguiente disminución del coste de mantenimiento y evolución) de diseño e implantación de dicha personalización respecto al resto de la aplicación. Este rasgo es

especialmente deseable dada la alta frecuencia de cambio de estas estrategias para adaptarse a las condiciones variables del entorno.

La primera aproximación ha sido la adoptada por aproximaciones de modelado de aplicaciones adaptivas bien conocidas como es el caso de UWE (Koch, 2000), mientras que la externalización de políticas ha sido adoptada en aproximaciones como W3I3 (Ceri *et al.*, 2000). En OO-H se contemplan ambas posibilidades. En primer lugar, OO-H permite especificar reglas de personalización en base a filtros asociados a eventos de activación de enlaces (de manera análoga al resto de restricciones del sistema), lo cual produce su soporte implícito en el método. Además, OO-H define un mecanismo de definición de reglas en base a una plantilla XML que, apoyado por una extensión de la arquitectura que incluye un motor de reglas, permite su externalización (Garrigós *et al.*, 2002).

Como inconveniente del uso de reglas podemos citar la necesidad de establecer mecanismos de resolución de conflictos en caso de contradicción entre las reglas.

Una clasificación exhaustiva de los distintos tipos de personalización, atendiendo a la diferenciación entre técnicas de personalización de contenido, de navegación y de presentación, puede ser consultada en **??**.

7.1.3 Riesgos de la personalización

Además de las ventajas e inconvenientes propios de cada técnica, existen numerosos aspectos que hay que tener en cuenta a la hora de decidirse por una u otra forma de personalización, que incluyen consideraciones relativas a (1) los costes (en términos tanto monetarios como de eficiencia y escalabilidad), (2) estrategias de privacidad y seguridad de los datos, (3) control de la propia adaptación, en términos de quién la autoriza, en qué momento se realiza y cómo se puede deshacer en caso de no adecuarse a las necesidades del usuario y (4) consideraciones legales y éticas.

En primer lugar, hay que evaluar el coste monetario asociado a determinadas estrategias, que a menudo requieren el uso de complejos algoritmos de minería de datos (*Data Mining*). Por otro lado es necesario evaluar el tiempo de respuesta de las aplicaciones personalizadas ante incrementos en el número de usuarios. Este parámetro es fundamental para determinar aspectos como la granularidad de la información analizada, el método y la frecuencia de análisis de datos o incluso cuestiones relativas a la arquitectura, como es el uso de servidores de caché.

Además, debido al tipo de información manejada, es necesario asegurar al usuario que el conocimiento (explícito o implícito) que tiene el sistema sobre él no va a ser accedido, difundido ni utilizado para otros fines que el facilitar y agilizar la tarea del propio usuario en el sistema. En este sentido existen estándares de seguridad e intercambio de datos como son el CPEX (*Customer Profile EXchange* (Customer Profile EXchange, 2001)) o el P3P (*Platform for Privacy Preferences Project* (W3C, 2000)) que pueden ser integrados para la consecución de este objetivo.

Otro aspecto importante es la especificación del modo en que se va a controlar la adaptación dinámica (adaptividad y proactividad) de la interfaz, si es que existe. De hecho la adición de cierto grado de inteligencia a la aplicación en detrimento del control que el usuario tiene sobre ella no sólo es contrario a la propia filosofía del paradigma hipertextual, sino que además disminuye el grado de estabilidad de las interfaces, lo cual dificulta la orientación del usuario en el sistema (Koch, 2001). Con el fin de evitar este problema, hay que evaluar cuidadosamente cuestiones como (VanderMeer *et al.*, 2000):

- Qué hacer cuando se infiere una nueva acción de personalización: ¿debe ser validada por un administrador o por el contrario debe ser ejecutada de manera automática?
- En qué orden se deben ejecutar las distintas acciones de personalización y qué hacer si existe una contradicción.

- Cuándo materializar la acción de personalización: inmediatamente o la próxima vez que el usuario entre al sistema.
- Período de validez de dicho cambio: a largo plazo (a nivel de aplicación) o a corto plazo (a nivel de sesión).
- Qué mecanismos se deben ofertar para que el usuario pueda corregir o evitar una personalización no deseada. Cuando registramos la actividad de un usuario en el sistema no es posible asegurar que dicho usuario no esté actuando de manera atípica. Las aplicaciones personalizadas deben por tanto en esos casos ser capaces de detectar esta atipicidad con el fin de eliminar el efecto de personalización causado por el conocimiento previo con carácter temporal (para la sesión actual, de manera que no interfiera en la ejecución de esta actividad atípica) o permanente (para cualquier sesión posterior). Aún más, intentar adivinar al usuario incluye un grado de impredictibilidad en la aplicación (cambios que se producen sin que el usuario aparentemente haya hecho nada) que hay que valorar cuidadosamente, ya que puede causar confusión.

Por último, no podemos olvidar que la personalización presenta preguntas legales y éticas, al permitir la identificación individual del usuario y sus preferencias. Es necesario por tanto respetar las restricciones impuestas por la legislación vigente antes de diseñar la política de personalización de la aplicación que queremos implantar.

Todos estos problemas, unidos a la sobrecarga que las estrategias de personalización introducen en el sistema, causan que la mayoría de las respuestas personalizadas se hayan basado hasta hace relativamente poco tiempo en un análisis *off-line* de ficheros de traza[4] o en técnicas de perfil estáticas y con respuestas enlatadas (VanderMeer *et al.*, 2000). Esta tendencia sin embargo está cambiando rápidamente, en parte gracias al éxito de estrategias de personalización dinámicas como las implantadas en portales como Yahoo o aplicaciones como Amazon. En cualquier caso, e independien-

[4] *log-files*

temente de la estrategia adoptada, es importante recordar que el no tener en cuenta las cuestiones anteriormente planteadas puede causar que la personalización se convierta en un obstáculo y no una ventaja para que el usuario cumpla sus objetivos en el sistema.

7.1.4 Clasificación de aplicaciones personalizadas

La combinación de técnicas de adquisición y de personalización[5] define la capacidad de adaptación de la aplicación a las condiciones de usuario, de sistema y/o de contexto de interacción, dando lugar a su clasificación en tres grandes grupos: adaptables, adaptivas o proactivas (Kappel *et al.*, 2000; Wu, 2001; Fraternali & Paolini, 2000), tal y como podemos observar en la Fig. 7.1.

Fuente Información / Causa personalización	*Información proporcionada explícitamente por el usuario o por el diseñador*	*Información inferida a partir de la actividad del usuario o del contexto de uso de la aplicación*
Personalización causada por la interacción explícita usuario-sistema	ADAPTABLES	ADAPTIVAS
Personalización causada por evento ajeno a la interacción explícita usuario-sistema	PROACTIVAS	

Figura 7.1. Tipos de aplicaciones personalizadas

Las **aplicaciones adaptables**[6] se caracterizan por un modelo de adquisición basado exclusivamente en información introducida por el diseñador o por el propio usuario de manera explícita en el sistema. Además, la acción de personalización también es reactiva, es decir, se produce ante una acción explícita de dicho usuario en el seno de la aplicación (e.g. invocar una nueva página). Ejemplos típicos de este tipo de aplicaciones son las que incluyen un formulario de preferencias cuya cumplimentación causa una selección del tipo de información presentada, reordenación de menús, cambios de colores y letras, etc.

[5] A las técnicas de personalización algunos autores las denominan *técnicas de adaptación* (Koch, 2001).

[6] *Customizable* (Kappel *et al.*, 2000)

Las **aplicaciones adaptivas** se diferencian de las adaptables en que durante el proceso de adquisición recogen y analizan información acerca de la actividad del usuario en el sistema y/o su entorno de interacción (no necesariamente relacionada con la introducción de datos por parte de dicho usuario). Como las adaptables, son aplicaciones en las que las acciones de personalización se producen siempre ante alguna acción del usuario en el sistema. Ejemplos típicos de este tipo de aplicaciones son los sistemas de recomendación utilizados como apoyo en numerosas aplicaciones de comercio electrónico.

Por último se suele hablar de **aplicaciones proactivas** (Fraternali & Paolini, 2000) para referirse a aquellas aplicaciones que son capaces de detectar *de motu propio* cambios en el entorno de interacción (ya sean características del usuario, del sistema o del contexto) y reaccionar en consecuencia, actuando de este modo como *observadoras* (Gamma *et al.*, 1995) de dicho entorno. Su principal diferencia respecto a las aplicaciones reactivas (adaptables o adaptivas) es que en las reactivas cualquier acción de personalización se produce como *reacción* ante un evento del usuario cuya vista va a ser personalizada (e.g. activación de un enlace, introducción de un determinado valor, etc.), y a partir de ese momento la página permanece inalterable aunque cambien las condiciones que dieron lugar a dicha personalización. Por el contrario, en una aplicación proactiva la aplicación puede cambiar la vista que el usuario tiene del sistema sin que éste realice ninguna acción (e.g. por el transcurso de un lapso de tiempo, por la entrada de nuevos usuarios en el sistema, por un cambio en el estado de la aplicación, etc.). Implica por tanto un giro hacia un esquema *push*, frente al tradicional esquema *pull* de las aplicaciones web. Un ejemplo de comportamiento proactivo es la capacidad por parte de la aplicación de detectar un cambio en localización de un dispositivo de acceso móvil, y la modificación automática de la vista de información ofertada en función de dicha localización (sin necesidad de que el usuario realice ninguna acción).

Una vez identificado el problema, a continuación presentamos las respuestas que, tanto desde el ámbito de la comunidad hiperme-

dial como de otras áreas, se han propuesto, y que están en la base de nuestra aproximación.

7.2 Personalización e Hipermedia

El reto de la personalización, lejos de ser una innovación de la comunidad hipermedial, ha sido un tema de discusión desde que se empezó a considerar al usuario como eje alrededor del cual debía girar el desarrollo de aplicaciones interactivas. Numerosos grupos de investigación provenientes de áreas tan dispares como la Inteligencia Artificial o la Interacción Usuario-Máquina han presentado incontables propuestas (Avery & Zeckhauser, 1997; Bra, 1999; Carroll & Aaronson, 1988; McIlhagga *et al.*, 1998) que, haciendo uso de tecnologías diversas como son las bases de datos, *cookies*, generación dinámica de páginas, algoritmos de aprendizaje computacional, técnicas de descubrimiento de patrones, inferencia basada en reglas o minería de datos, son capaces de desarrollar interfaces inteligentes, preparadas para predecir el comportamiento del usuario en el sistema y facilitar la consecución de sus objetivos individuales (ACM, 2000; Kappel *et al.*, 2000).

Sin embargo la aparición del hipertexto ha dado un nuevo sentido a esta personalización. En este tipo de aplicaciones, el requisito de fidelidad del usuario ha provocado un esfuerzo adicional para cubrir las necesidades particulares de estructuración navegacional, así como para dar soporte a contextos de uso altamente volátiles en términos de disparidad de dispositivos, imprevisibilidad de la red o ubicuidad y atemporalidad del usuario. A nivel empresarial este esfuerzo se ha materializado en la aparición de nuevos consorcios (ver e.g. el Consorcio de Personalización (Personalization Consortium, 2002)) y estándares para la descripción tanto de preferencias de usuario y capacidades de dispositivo (ver CC/PP (W3C, 2000)) como para asegurar el intercambio, la privacidad y seguridad de la información que posibilita la personalización (ver e.g. CPEX o P3P). Además, se ha desarrollado un gran número de herramientas y paquetes comerciales (e.g. ILog JRules, WebSphe-

re, Rainbow, WebPlaces, LikeMinds o Velocity, por citar algunas) que facilitan el uso de las técnicas y estrategias de personalización y que por tanto actualmente dan soporte a numerosas aplicaciones web personalizadas.

Con todas estas propuestas, el principal problema con el que nos encontramos en la actualidad es el bajo nivel de abstracción al que se está produciendo la integración de estos estándares y herramientas en las aplicaciones actuales para dar soporte a las políticas de personalización deseadas. Este hecho está causando problemas de falta de reuso y dificultad de mantenimiento y escalabilidad de las aplicaciones personalizadas resultantes.

Las propuestas hipermediales han sido desde sus orígenes conscientes de la importancia de este problema, y así propuestas pioneras como HDM (Garzotto *et al.*, 1993) incorporan desde sus primeras versiones mecanismos como las perspectivas, con el fin de intentar dar una respuesta de diseño (mediante la técnica de variación de contenido (Koch, 2001)) a un problema de adaptación de contenido de la aplicación. Estas soluciones particulares se han ido generalizando y sistematizando con la madurez de estas propuestas de desarrollo hipermedial. Así, a nivel teórico, se ha avanzado en la clasificación de diversos métodos y técnicas para la personalización de contenido, navegación y presentación (Koch, 2001). Estos estudios a su vez han facilitado la inclusión en propuestas como (Ceri *et al.*, 2000; Koch & Wirsing, 2001; Rossi *et al.*, 2001) o el propio OO-H de técnicas y marcos específicos de modelado de personalización que faciliten su posterior implantación y mantenimiento.

El planteamiento de estos mecanismos no es trivial ya que, como ya comentamos en el capítulo 3, la inclusión de políticas de personalización en el marco de métodos hipermediales afecta a todos los ejes de modelado de una aplicación web (ver Fig. 7.2 (Retschitzegger & Schwinger, 2000)).

Esta realidad aún se complica más si tenemos en cuenta que, en la actualidad, algunos aspectos de las políticas de personalización son imposibles de detectar antes de la puesta en marcha de la

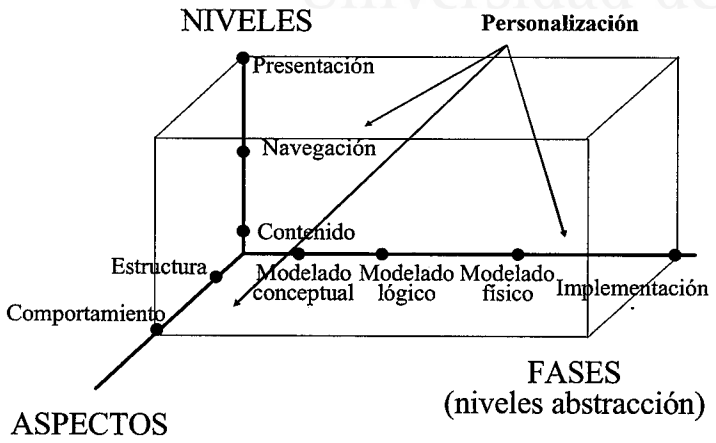

Figura 7.2. Influencia de la Personalización en las dimensiones de modelado de las aplicaciones web

aplicación, lo que a su vez obliga a estos métodos a incluir mecanismos que posibiliten externalización de políticas y el soporte a su cambio una vez implantada la aplicación. Soluciones de estas características han sido ofrecidas en el seno de propuestas como (Ceri *et al.*, 1999). El manejo adecuado de éstos y otros aspectos hace difícil su inclusión eficiente en las aplicaciones sin el uso de técnicas que aumenten el nivel de abstracción al que se opera actualmente.

En este sentido, el uso de modelos de análisis y diseño como los presentes en la mayor parte de las propuestas hipermediales proporciona una oportunidad única de plantear estrategias de personalización basadas en la semántica asociada a los átomos de información y no en su estructuración física. Como ejemplo, la mayoría de modelos de diseño de navegación que presentamos en el capítulo 2 permiten distinguir entre estructuras de acceso y nodos de información. Esta distinción permite que, ante una estra-

tegia como es la generación de atajos a las páginas más visitadas, se pueda determinar qué páginas materializan esas estructuras de acceso y por tanto pueden ser excluidas de esta lista de atajos.

Además, el modelado y diseño de la personalización dentro de un marco conceptual permite entender mejor los mecanismos utilizados, y descubrir rasgos comunes que posibilitan el reuso de patrones, componentes, algoritmos o incluso subsistemas (Rossi *et al.*, 2001).

7.3 El papel de la personalización en OO-H

OO-H, concebido desde sus orígenes como un método de desarrollo de aplicaciones hipermediales orientado al usuario (característica fundamental para facilitar la definición de estrategias de personalización (Kramer *et al.*, 2000)), vertebra su soporte a la integración de políticas de personalización en torno a tres elementos fundamentales (Cachero *et al.*, 2002c):

- Un modelo de usuario implícito que gira en torno a los conceptos de *usuario* y *rol*. El enriquecimiento de este modelo de usuario durante la actividad de diseño de dominio supone la inclusión de nuevas estructuras y relaciones que reflejan las características de cada usuario/rol particular.
- Un modelo de referencia extensible que incluye un conjunto de mecanismos de registro y análisis de la actividad del usuario en el sistema, y permite la adaptación dinámica de las distintas vistas de la aplicación en función de esta información.
- Un conjunto de tareas que permiten modelar los aspectos relacionados con la personalización de contenido, navegación y presentación durante el proceso de diseño de la aplicación.

De ellas, el modelo de usuario y el modelo de referencia son el soporte básico para la definición de políticas de personalización no triviales (Koch, 2001), mientras que el proceso de diseño define el modo en que estos modelos son utilizados como base para

modificar el resto de modelos de OO-H, con el fin de reflejar los requisitos de personalización.

La aportación más importante de OO-H en este sentido es precisamente este proceso de diseño, que no sólo favorece la recogida de requisitos de personalización desde las primeras fases del método, sino que integra de manera transparente el proceso de modelado de estos requisitos. Para ello OO-H apuesta por la homogeneización de las técnicas de modelado de la personalización mediante el uso de *Filtros de adquisición* y *Filtros de personalización*[7], cuya única diferencia respecto a los filtros tradicionales es que incluyen referencias a elementos del modelo de usuario y/o del modelo de referencia de OO-H. Este rasgo no sólo facilita el proceso de aprendizaje de los diseñadores, sino que está simplificando en gran medida la implementación de su soporte en el entorno de desarrollo de OO-H.

A continuación profundizaremos en cada uno de estos elementos.

7.4 El modelo de usuario en OO-H

El modelo de usuario se materializa en OO-H en un Diagrama de Clases que complementa el modelo de dominio de la aplicación, y captura información acerca de las características que el sistema particular que estamos modelando cree que posee el usuario. Estas características pueden incluir su conocimiento acerca del dominio, su bagaje, sus intereses y preferencias, su experiencia y sus tareas y metas en la aplicación. Esta información no tiene por qué coincidir con lo que opina el diseñador ni con las características que realmente posee el usuario. De hecho un modelo incompleto e inexacto puede todavía ser útil para mejorar características como la velocidad operacional o la satisfacción del usuario en el sistema (Koch, 2001).

Este modelo de usuario, que debe ser definido en función de los requisitos de personalización que deseemos soportar en una apli-

[7] Estos conceptos serán detallados en la sección 7.6.6

cación particular, se vertebra en torno a los conceptos de *usuario* y *rol*, al igual que ocurre en otras aproximaciones hipermediales (Ceri *et al.*, 1999). Para ello, OO-H proporciona un modelo de usuario mínimo que, partiendo de la clase *Usuario* proporcionada por el modelo de referencia de OO-H[8], define una jerarquía de herencia entre ésta y todas las clases de dominio estereotipadas como <<*actor*>>, que son las que definen los posibles roles de un usuario en la aplicación. De este modo, se dota a todos los roles de las capacidades de personalización proporcionadas por OO-H a través de esta clase *Usuario*.

Este modelo básico (formado por la clase *Usuario* y la jerarquía de herencia correspondiente a los distintos *actores* del sistema) puede enriquecerse con las clases, atributos y relaciones necesarias para dar soporte a la política de personalización deseada. El modo en que se construye este modelo de usuario en OO-H será visto en la sección 7.6.5.

7.5 Soporte a la personalización en el modelo de referencia de OO-H

Además del modelo de usuario, la inclusión de políticas de personalización operativas a menudo requiere el análisis de la actividad del usuario en el sistema, actividad que no siempre causa un cambio en el estado de la aplicación. Es por ello que OO-H incorpora como parte de su modelo de referencia un conjunto de estructuras y operaciones que dan soporte a las posibles condiciones y acciones de personalización contempladas en OO-H. Una visión parcial de estas estructuras y sus interrelaciones puede ser visto en la Fig.7.3 .

Tal y como podemos observar mediante la comparación de las Figs. 7.8 y 7.3, este modelo está relacionado con el modelo de usuario a través de la clase *Usuario* que comentábamos en la sección anterior, y refleja las dos posibles fuentes de información

[8] Este modelo de referencia será explicado en la sección 7.5.

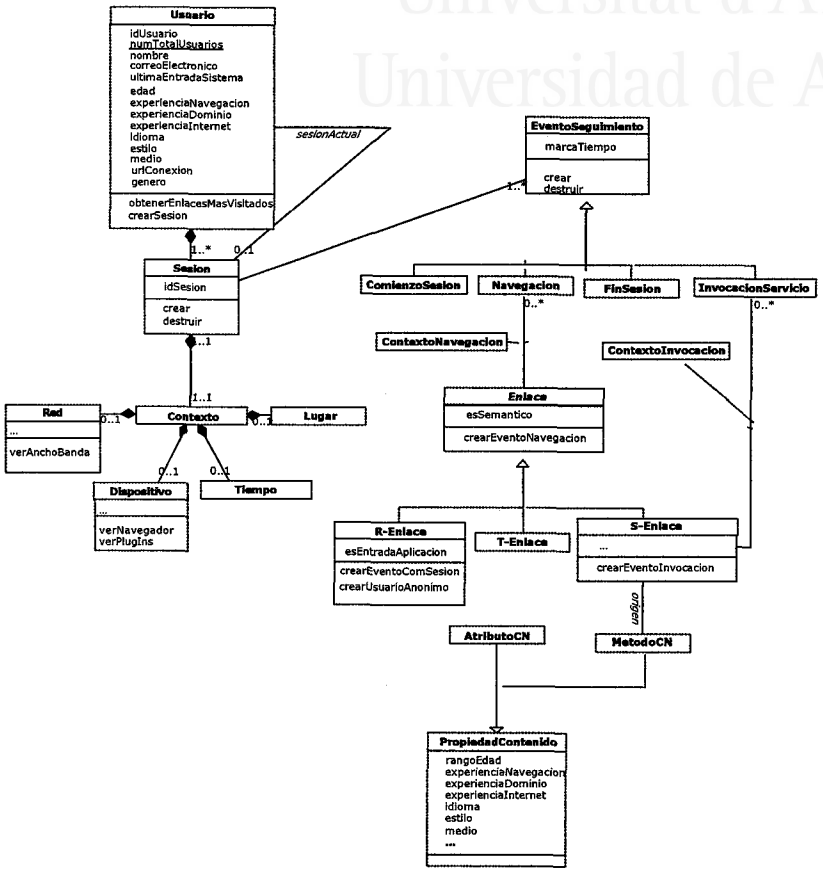

Figura 7.3. Modelo de Referencia de OO-H: vista de personalización

que intervienen en el modelado de la personalización en OO-H, y que son (1) características y preferencias de usuario y (2) contexto de interacción, que a su vez incluye tanto información acerca de la actividad del usuario en el sistema como información relativa a condiciones físicas de la interacción (ubicación, momento de interacción, etc.).

7.5.1 Características y preferencias del usuario

En la Fig. 7.3 podemos observar cómo en primer lugar las características y preferencias de usuario se almacenan en la clase *Usuario*, que en el estado actual de desarrollo de OO-H incluye datos relativos a la edad, género, URL de conexión, idioma, estilo (abreviado, simplificado, experto), medio (texto, imagen, vídeo), experiencia de navegación, etc. Estas características son la base para la implementación de una *personalización de contenido* mediante la técnica de *fragmentos condicionales* (Koch, 2001). Para ello OO-H incluye una clase *PropiedadContenido* que asocia a cada atributo/método visualizado en el diagrama navegacional un conjunto de propiedades (*rangoEdad*, *idioma*, *estilo*, *medio*, etc.) que deben ser un subconjunto de las definidas para la clase *Usuario*. La implementación de esta técnica causa que, cuando la aplicación detecta que debe visualizar algún atributo (*AtributoCN*) o acceso a método (*MetodoCN*), antes se compruebe que sus *PropiedadesContenido* coinciden con las especificadas para el usuario que está accediendo a la aplicación. Por ejemplo, supongamos que el atributo *descripcion* de la clase *Hotel* es un atributo multivaluado donde el primer valor corresponde a una perspectiva en castellano y el segundo a una perspectiva en inglés. Si este atributo aparece en el modelo de navegación de algún usuario, y este usuario tiene como preferencia de idioma el castellano, sólo se mostrará el primer valor, y se ocultará por tanto la descripción en inglés. Los atributos/métodos que no tengan definidos valores para alguna de sus propiedades se muestran independientemente del valor que el usuario que esté interactuando con la aplicación tenga determinado para esa característica.

7.5.2 Actividad del usuario en el sistema

Por otro lado, la actividad del usuario en el sistema puede abarcar aspectos muy diversos: búsquedas realizadas, páginas visitadas, productos o servicios adquiridos, etc. Es importante hacer notar que, aunque esta información puede ser proporcionada

explícitamente por el propio usuario mediante la definición de un simple formulario de entrada, es un hecho reconocido que este mecanismo es un método invasivo debido a que requiere un cambio en el foco de atención del usuario, lo que a su vez causa que en la mayoría de los casos éste no personalice (VanderMeer *et al.*, 2000). Es por ello que actualmente la tendencia generalizada es intentar deducir el mayor número de características a partir de información que puede ser obtenida de manera transparente al usuario.

Para ello, en OO-H cada entrada al sistema se materializa en una nueva instanciación de la clase *Usuario* (ver Fig 7.3), que inmediatamente instancia una nueva *Sesion*. Si el usuario se identifica en el sistema, se destruye este usuario anónimo (y su correspondiente sesión), y se reactiva el objeto correspondiente al usuario identificado (que es persistente). A este objeto se le asigna entonces una nueva sesión, que se añade al conjunto de sesiones anteriores del usuario en el sistema.

Los objetos de tipo *Sesion* guardan información acerca de cuatro tipos de eventos de usuario en el sistema (*EventoSeguimiento*) que son:

- Comienzo de Sesión: implica la entrada del usuario en el sistema.
- Navegación: implica la activación por parte del usuario de un enlace de navegación en la interfaz asociada. Una acción de navegación guarda su contexto, es decir, las condiciones de filtro que tenía asociadas cuando se activó.
- Invocación de Servicio: implica la invocación de un servicio ofertado por la lógica. Este tipo de acción supone guardar tanto el enlace de servicio activado como los valores introducidos en los parámetros para esa invocación.
- Fin de Sesión: implica la salida del usuario del sistema.

Independientemente de su tipo, todos los *EventoSeguimiento* llevan asociada una *marca de tiempo* que determina el orden en que se han ido ejecutando las acciones.

La creación de nuevos ítems de seguimiento y su asociación a la sesión actual del usuario se realiza mediante la asociación de servicios de creación de objetos de tipo *EventoSeguimiento* a los distintos tipos de enlace que ofrece OO-H:

- La activación del enlace de entrada a la aplicación lleva en OO-H implícitamente asociada la creación de un ítem de seguimiento de tipo *ComienzoSesion*.
- La activación de un enlace de servicio como puede ser el *Crear Nueva Reserva* crea implícitamente un objeto de seguimiento de tipo *InvocacionServicio*, que se asocia al *S-Enlace* en cuestión y a la sesión actual del usuario que lo ha activado. Esta invocación guarda además información acerca del *ContextoInvocación*: valor de los parámetros, valor de los posibles filtros asociados al enlace de servicio, etc.
- La acción de activar cualquier tipo de *Enlace* de navegación en OO-H provoca, paralelamente al efecto de navegación, la creación de un objeto de seguimiento de tipo *Navegacion*, que nuevamente almacena, además de una relación con el objeto que representa al enlace en cuestión, la información referente al *ContextoNavegacion*: población origen y destino del enlace, valor de los posibles filtros, etc. OO-H también guarda una referencia a su carácter semántico[9].
- Por último, la instanciación de un objeto de seguimiento de tipo *FinSesion* (que, como efecto colateral, finaliza la sesión del usuario en el sistema) se produce de manera automática cuando la aplicación detecta un período de inactividad en el sistema superior a un valor umbral (en este momento establecido en cinco minutos).

Esta información sirve de base para la inclusión de diversos algoritmos que permiten dar respuesta a las diversas estrategias de personalización. Por ejemplo, en la sección 7.6.6 veremos cómo el servicio *Usuario.obtenerEnlacesMasVisitados(boolean semantico,*

[9] Esta característica de los enlaces fue presentada en el capítulo 5.

in num, out listaEnlaces) proporcionado por el modelo de referencia (ver Fig. 7.3) permite de una forma muy sencilla la definición de una colección dinámica que incluya los enlaces más visitados en función del usuario.

Los ítems de seguimiento son especialmente útiles para la detección de nuevos patrones de navegación de usuario (Mobasher *et al.*, 1999). Además, y debido a que en OO-H los ítems de seguimiento guardan referencias a enlaces del modelo de navegación (ver Fig. 7.3), una posible categorización exhaustiva de estos enlaces (por ejemplo identificación de determinados enlaces de servicio como enlaces de tipo 'compra', o de determinados enlaces de navegación como enlaces de tipo 'descarga' (Bieber *et al.*, 1999)) permitiría definir estrategias mucho más precisas, que en OO-H están contempladas como trabajos futuros.

7.5.3 Información de contexto

Por último, el marco de referencia de OO-H ofrece un conjunto de estructuras para el almacenamiento del contexto físico en el que se produce la interacción usuario-aplicación. Este contexto se incluye como parte de la sesión del usuario, y en OO-H se vertebra en torno a cuatro ejes:

- Contexto de red: latencia, ancho de banda etc. de la conexión.
- Contexto Temporal: fecha y hora local de la conexión.
- Contexto de Dispositivo: tipo de dispositivo (portátil, PC, etc.) y software instalado en él.
- Contexto de Localización: ubicación del cliente.

La inclusión en el modelo de referencia de las clases correspondientes y de los servicios necesarios para poder consultar estas características de contexto (siempre que el entorno de implementación lo permita) enriquece el tipo de políticas de personalización que pueden ser implementadas.

Como ejemplo, la Fig. 7.3 muestra cómo OO-H materializa la adquisición de información acerca del ancho de banda actual de la conexión mediante la inclusión de un servicio *verAnchoBanda*, así como consultar el navegador y *plug-ins* que tiene instalados el cliente mediante la invocación de los servicios *verNavegador* y *verPlugIns*. De este modo, podríamos por ejemplo especificar en cualquier aplicación la utilización del criterio de aumento del ancho de banda de la conexión como condición de personalización (e.g. para proporcionar la posibilidad de visualizar panoramas, vídeos, etc. si el ancho de banda supera cierto valor umbral).

7.6 Personalización y proceso de diseño

El tercer elemento alrededor del cual OO-H vertebra su soporte a la integración de políticas de personalización es su proceso de diseño. Como ya vimos en la sección 7.2, los requisitos de personalización pueden influir en cualquier fase, nivel y/o aspecto de una aplicación web. Es por ello que es importante tenerlos en cuenta durante la ejecución de todo el flujo de actividades de OO-H[10].

Antes de estudiar los efectos que la inclusión de una política de personalización tiene sobre estas actividades, y con el fin de convertir nuestro Sistema de Gestión de Hoteles en una aplicación personalizable sobre la que ejemplificar los distintos conceptos, a continuación extenderemos la descripción del sistema.

7.6.1 Ampliación del Caso de Estudio: Un Sistema de Gestión de Hoteles personalizado

Supongamos que durante la fase de análisis de requisitos se ha identificado un conjunto de características que se desea que varíen en función de las preferencias individuales de cada usuario:

- Los tipos de habitación sobre los que cada cliente realiza reservas pueden estar restringidos por las preferencias de dicho clien-

[10] Dicho flujo de actividades fue presentado en el capítulo 3

te. Estas preferencias deben ser definidas por el propio usuario mediante la cumplimentación de un formulario de inicio.

- La aplicación debe ofertar un menú de acceso rápido a las tres tareas (requisitos funcionales) más demandadas por cada usuario.
- Cualquier usuario debe poder configurar el tipo, tamaño y color de fuente, el color de los enlaces y el color de fondo de la presentación de la aplicación.

El tipo de personalización incluida en este ejemplo o bien es reversible (tipos de habitación o colores) o bien no supone ningún problema para el usuario aunque no sea correcta: la construcción de un menú adicional de acceso rápido puede no ser de ayuda para un usuario en un momento determinado, pero en cualquier caso no va a interferir en su trabajo en el sistema. Es por ello que no ha sido necesario introducir ningún requisito adicional que permita la reversibilidad del efecto de una personalización no deseada.

A continuación veremos cómo se va introduciendo el modelado de estos requisitos en las distintas actividades del método.

7.6.2 Personalización y análisis de requisitos funcionales

El proceso de modelado de aplicaciones personalizadas comienza con la captura de requisitos que afecten a la funcionalidad que la interfaz debe ofrecer a cada usuario.

En nuestro ejemplo, estos requisitos son dos: mantener las preferencias de tipos de habitación (sólo para los clientes) y mantener las preferencias de presentación (todos los usuarios). Estos requisitos se materializan en sendos casos de uso, que se muestran sombreados en la Fig. 7.4. En ella vemos cómo el caso de uso *Mantener Tipos Habitación* se asocia directamente al actor *Cliente*, que tiene además, como necesidades particulares, el *Ver Facturas* y *Realizar Nueva Reserva*, tal y como comentamos en la sección 4.2. Por otro lado, el caso de uso *Mantener Preferencias Presentación* se asocia al actor *Anónimo*, que es el rol que

cualquier usuario adquiere en la aplicación por el simple hecho de entrar a ella. La funcionalidad de este rol, y por tanto también el requisito *Mantener Preferencias Presentación* (ver Fig. 7.4) es heredada tanto por el *Recepcionista* como por el *Cliente*.

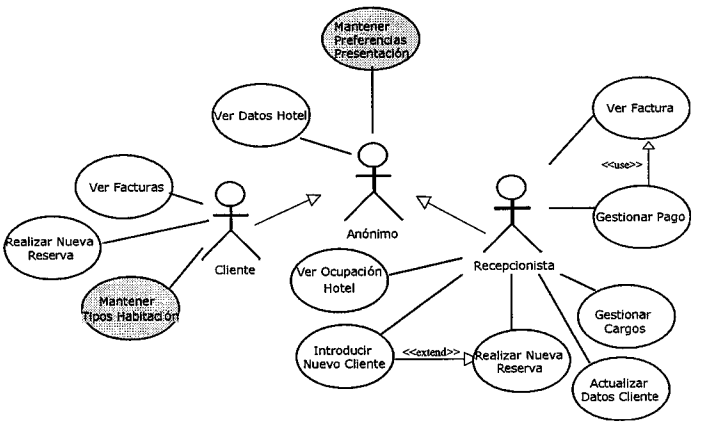

Figura 7.4. Vista parcial del DCU correspondiente al SGH personalizado

7.6.3 Personalización y análisis de dominio

Una vez obtenidos los casos de uso, el resultado de analizar los requisitos de personalización de la aplicación durante la fase de análisis de dominio produce un refinamiento del diagrama de clases de análisis (DCA) resultante, cuyo efecto más representativo es la inclusión de la clase genérica *Usuario*, de la que heredan todas las clases que representan roles de usuario en el dominio.

En nuestro ejemplo (ver Fig. 7.5) observamos cómo se ha introducido esta clase *Usuario* (que en este caso es suficiente para soportar al rol de usuario anónimo) y la jerarquía de herencia con las clases *Recepcionista* y *Cliente*. La clase *Usuario* es la base sobre la que se realiza el refinamiento durante la actividad de diseño de dominio para incluir los atributos y métodos necesarios para soportar la política de personalización deseada, tal y como veremos en la sección 7.6.5.

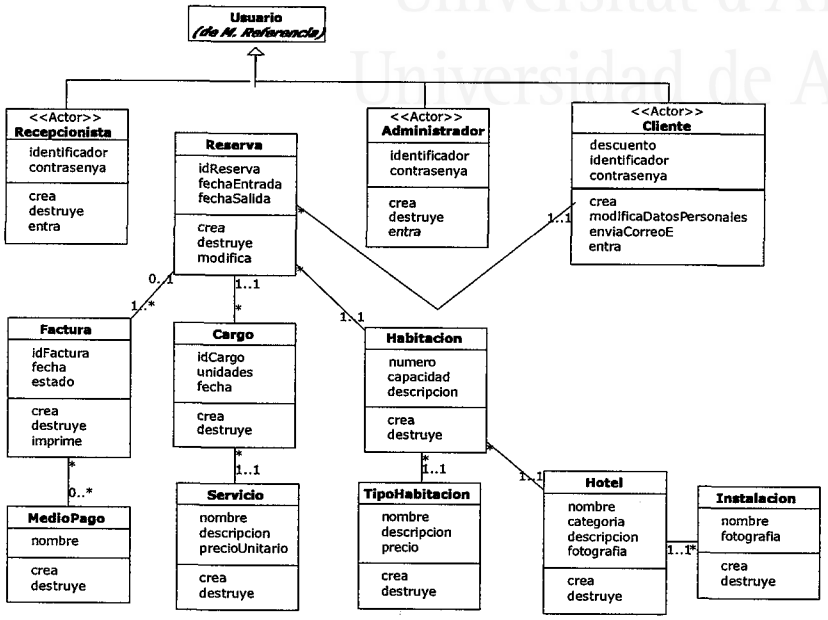

Figura 7.5. Vista parcial del DCA correspondiente al SGH personalizado

Esta jerarquía de usuarios es el primer paso hacia la construcción del **Modelo de Usuario** que presentamos en la sección 7.4. Dentro de este modelo, la clase *Usuario* constituye el nexo de unión del modelo de usuario con el modelo de referencia de OO-H, tal y como vimos en la sección 7.5.

7.6.4 Personalización y análisis de navegación

Además de las modificaciones en el diagrama de clases de análisis, la inclusión de nuevos requisitos de personalización puede causar también, como ya vimos en el capítulo 5, la inclusión de nuevos enlaces semánticos y nuevas unidades semánticas que dan respuesta a estos requisitos a nivel de análisis de navegación, con el fin de reestructurar la organización genérica de la interfaz en base a las nuevas tareas de personalización.

En nuestro ejemplo (ver Fig. 7.6) podemos observar cómo en el diagrama de navegación semántica del *Recepcionista* se ha incluido un nuevo enlace semántico: *Mantener Preferencias Presentación*, que apunta a una nueva unidad semántica *Adaptación*. En el caso del diagrama de navegación semántica asociado al rol *Cliente* (ver Fig. 7.7), los enlaces semánticos introducidos son dos: el enlace *Mantener Preferencias Presentación* (común a ambos roles), y el enlace *Mantener Tipos Habitación*, que permite introducir y almacenar las preferencias de cada cliente con respecto a los tipos de las habitación en los que está interesado.

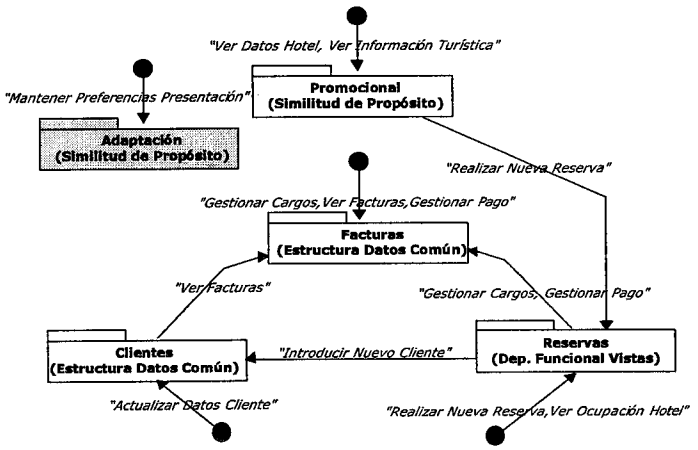

Figura 7.6. Vista parcial del DNS correspondiente al SGH personalizado vista Recepcionista

7.6.5 Personalización y diseño de dominio

La actividad de diseño de dominio es, junto a la actividad de diseño de navegación que veremos en la sección 7.6.6, una de las más afectadas por las consideraciones relativas a la personalización de la aplicación.

En efecto, la introducción de requisitos de personalización suele implicar la inclusión en el modelo de diseño de dominio de nuevas

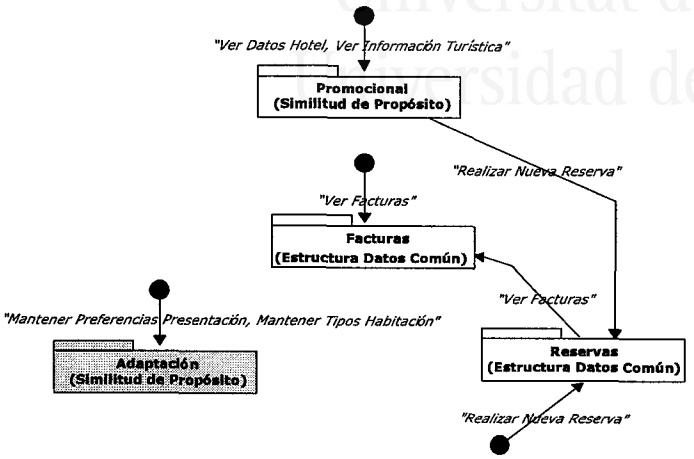

Figura 7.7. Vista parcial del DNS correspondiente al SGH personalizado vista Cliente

clases, relaciones, atributos, servicios, patrones de diseño, etc. que refinan y complementan las distintas clases del sistema.

La complejidad potencial de estos cambios sugiere la conveniencia de reflejar todos estos cambios de dominio en una nueva vista, el *Modelo de Usuario*, tal y como comentamos en la sección 7.4. Es por tanto durante esta fase de diseño de dominio cuando se construye el modelo de usuario de la aplicación, que constituye una de las características diferenciadoras de las aplicaciones personalizadas.

Volviendo a nuestro ejemplo (ver Fig. 7.8), la inclusión de los nuevos requisitos de personalización ha causado la creación de un modelo de usuario donde se ha conectado la clase *Usuario* a una nueva clase *Apariencia* que recoge las preferencias de presentación de cada usuario individual. Además, el almacenamiento de los tipos de habitación preferidos por cada cliente requiere la inclusión de una nueva relación *InteresTipoHabitacion* entre *Cliente* y *TipoHabitacion*, con una clase de asociación que almacena el valor

de dicho interés. Un posible modelo de usuario que incluye todos estos cambios puede ser visto en la Fig. 7.8 [11]

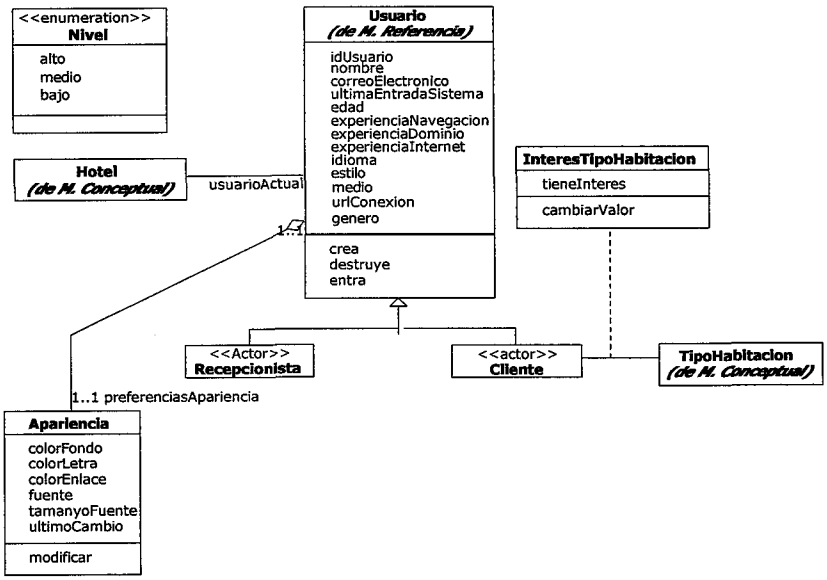

Figura 7.8. Modelo de Usuario del Sistema de Gestión de Hoteles

7.6.6 Personalización y diseño de navegación

Toda la información recogida en el modelo de usuario sirve de base sobre la que implementar la estrategia de personalización deseada. En OO-H el grueso de esta implementación se realiza sobre los modelos de navegación asociados a cada tipo de usuario de la aplicación, y consta de dos fases bien diferenciadas:

- Fase de adquisición, en la que se definen *Filtros de adquisición* que, asociados a los distintos enlaces del DAN, modelan el proce-

[11] Es interesante hacer notar cómo es posible, en este modelo de usuario, realizar una personalización sobre la propia personalización. Por ejemplo, podría introducirse un patrón *Strategies* (Gamma *et al.*, 1995), que permitiría definir una familia de algoritmos, encapsular cada uno de ellos, y convertirlos en intercambiables en función del usuario y/o de su contexto de interacción.

so de adquisición de información externa a la propia aplicación mediante la invocación de servicios del modelo de referencia, tal y como comentamos en la sección 7.5.3. Estos filtros de adquisición se complementan con la introducción, si es necesario, de nuevos enlaces de servicio que, asociados a los nuevos servicios introducidos en el *Modelo de Usuario*, modelan formularios de entrada, mediante los cuales el usuario puede introducir de manera explícita la información requerida. Estas dos técnicas, unidas al registro implícito por parte de las aplicaciones OO-H de la actividad del usuario en el sistema (en base a los eventos de usuario que comentamos en la sección 7.5.2) conforman la política de adquisición de OO-H.

- Fase de Personalización, en la que se definen *Filtros de Personalización* que materializan, en base a la información disponible, las acciones de personalización deseadas.

Es importante destacar cómo los *Filtros de adquisición* y los *Filtros de Personalización* sólo se diferencian de los filtros tradicionales de OO-H en que en su formulación se incluyen referencias a servicios y atributos del modelo de usuario de la aplicación y/o del modelo de referencia de OO-H.

Volviendo a nuestro ejemplo, en la Fig. 7.9 podemos ver las modificaciones en el modelo de navegación para el tipo de usuario *Recepcionista* causadas por los nuevos requisitos de personalización. El primer cambio consiste en la introducción de un nuevo destino navegacional *Adaptación*. En la explosión de este destino navegacional (ver Fig. 7.10), vemos cómo la clase navegacional *Apariencia*, que es una vista de la clase homónima contenida en el modelo de usuario, ha sido incluida en el modelo de diseño de navegación, de manera que ahora el recepcionista puede introducir los colores, fuentes, etc. que desea para personalizar su presentación. En cualquier momento, el recepcionista puede volver a este destino para cambiar estas preferencias, por lo que siempre mantiene el control sobre el cambio.

En la Fig. 7.10 también podemos observar cómo el R-Enlace *Mantener Preferencias Apariencia* tiene asociado un *Filtro de Perso-*

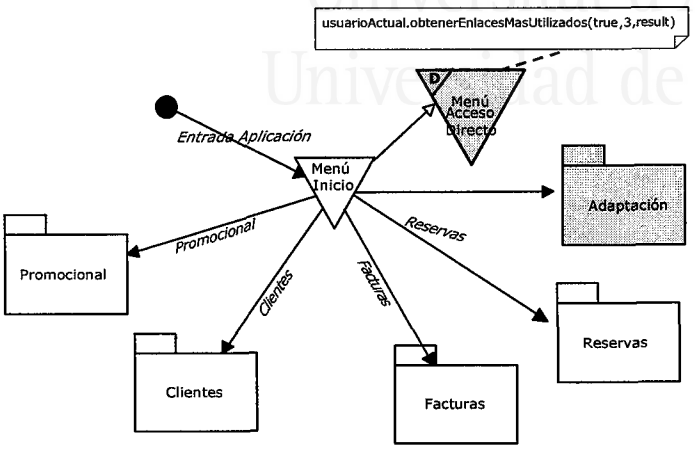

Figura 7.9. Primer Nivel de la Estructura Navegacional Personalizada del SGH vista Recepcionista

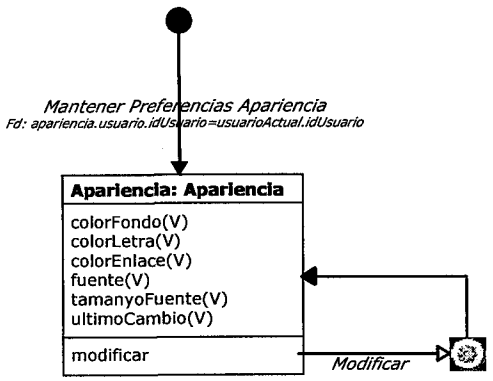

Figura 7.10. Explosión del DN Adaptación del SGH

nalización que, definido en destino, utiliza información contenida en el modelo de usuario referente al usuario actual del sistema. Para trabajar con dicho usuario actual, OO-H mantiene una referencia a este objeto (el usuario actual) en una variable del propio entorno de ejecución de la aplicación denominada *usuarioActual.*

Si ahora nos centramos en la gestión de *Nuevas Reservas* (ver Fig. 7.11), cuyo enlace de servicio requería la asignación de un valor

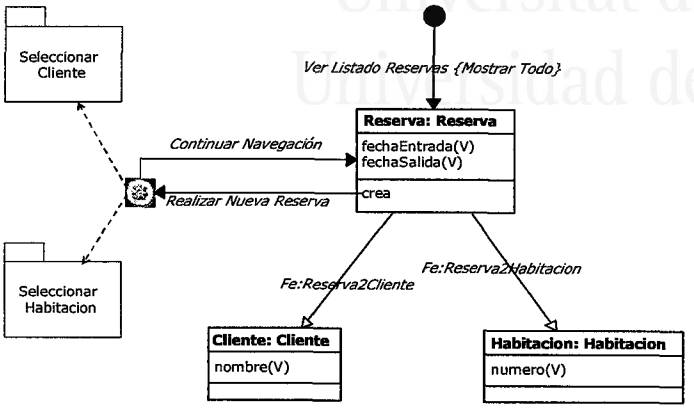

Figura 7.11. Enlace de servicio *Nueva Reserva*

a, entre otros parámetros, un parámetro *pCliente* y un parámetro *pHabitacion*, es posible personalizar las habitaciones entre las que el recepcionista puede elegir en función del cliente seleccionado. Para ello en el diagrama explosionado a partir del DN *Seleccionar Habitacion* debemos añadir un nuevo filtro de personalización a los enlaces que determinan el modo en que el recepcionista puede seleccionar la habitación a reservar, tal y como podemos observar en la Fig. 7.12[12].

Este filtro de personalización hace que el recepcionista sólo tenga acceso a los tipos de habitación para los que el cliente sobre el que se está efectuando la reserva ha establecido un *tieneInteres=true*, donde el atributo *tieneInteres* pertenece, como ya hemos comentado, a la clase *InteresTipoHabitacion* incluida en el modelo de usuario de la Fig. 7.8.

Por último, en la Fig. 7.9 también podemos observar cómo se modela en OO-H un requisito como es la generación de un menú personalizado. Este menú se define, como el resto de menús en OO-H, como una nueva colección *Menú Acceso Directo*. Una *D* en la esquina superior izquierda de esta colección indica que se trata de una colección dinámica, es decir, que los destinos

[12] Estos diagramas fueron explicados en la sección 6.3.2

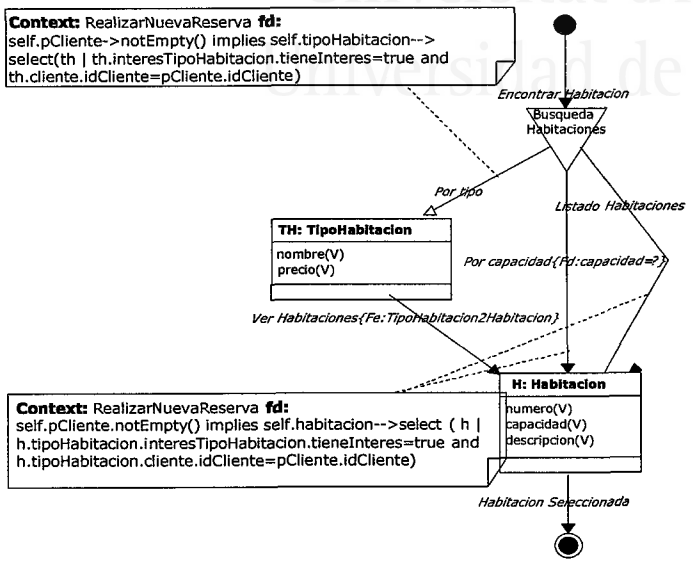

Figura 7.12. Primer Nivel de la Estructura Navegacional Personalizada del SGH

de los enlaces que agrupa no son conocidos en tiempo de diseño. Efectivamente, estos destinos se definen mediante un filtro de personalización que debe ser evaluado en tiempo de ejecución para averiguar las funciones más solicitadas por parte de cada usuario. Este filtro incluye una llamada al servicio *Usuario.obtenerEnlacesMasVisitados(boolean semantico, in num, out listaEnlaces)* proporcionado por el modelo de referencia (ver Fig. 7.3). Para materializar esta colección dinámica, basta con invocarlo con los parámetros *semantico=true* y *num=3*, y la aplicación nos devolverá los tres enlaces más frecuentados (presentes en más ítems de seguimiento) durante el conjunto de sesiones del usuario.

7.6.7 Personalización y diseño de presentación

Del mismo modo que OO-H permite la definición de distintas vistas del modelo de diseño de la navegación, una para cada tipo de usuario del sistema, OO-H también permite la definición

de distintas presentaciones para un mismo modelo de diseño de navegación, tal y como vimos en la Fig. 3.2.

Sin embargo, este mecanismo es demasiado engorroso para el modelado de requisitos de personalización a nivel de usuario individual. Con el fin de solventar este problema, OO-H permite la asociación de filtros de personalización a determinadas propiedades de presentación, cuyo valor pasa de este modo a ser calculado en tiempo de ejecución. Así, en nuestro ejemplo la manera de personalizar las presentación de las características contenidas en la clase *Apariencia* sería mediante la definición de un conjunto de fórmulas que hicieran referencia a los distintos valores de los atributos de esta clase, y asociar dichas fórmulas a la propiedad de presentación correspondiente. Estas propiedades de presentación serán comentadas en profundidad en el capítulo 8.

7.7 Conclusiones del capítulo

En este capítulo se ha proporcionado una visión general del concepto de *facilidad de uso* y el papel que la personalización de aplicaciones juega dentro de este concepto. A continuación hemos mostrado cómo la relevancia de este rasgo en las aplicaciones hipermediales ha provocado el interés de diversos grupos de investigación para incrementar el nivel de abstracción al que se definen estas políticas, y así conseguir aumentar su facilidad de mantenimiento.

OO-H comparte este interés, e incluye un conjunto de mecanismos que soportan la definición, como parte de su proceso de modelado de aplicaciones hipermediales, de una política de personalización que se integra en los distintos modelos que conforman el método. Para ello OO-H aboga por la construcción de un *modelo de usuario* que complementa el modelo de dominio y que representa las preferencias, características, etc. particulares del usuario relevantes para la definición de la política deseada. Además OO-H proporciona un *Modelo de Referencia* extensible que: (1) permite personalizar el contenido de la aplicación en función de un con-

junto de rasgos predefinidos, (2) dota a las aplicaciones de una capacidad de seguimiento de la actividad del usuario en la aplicación en base a cuatro tipos de eventos (entrada en el sistema, navegación, invocación de servicios y salida del sistema) y (3) permite la personalización de aplicaciones en función de variaciones en el contexto físico del usuario: localización, características del dispositivo, características de la red o características temporales de la conexión.

La definición a nivel de diseño navegacional de filtros de adquisición y filtros de personalización que hacen referencia a estas estructuras y se asocian a los distintos enlaces definidos durante la fase de diseño de navegación permite el modelado de aplicaciones personalizadas de una manera natural, similar a como se define el resto de restricciones de dicho modelo de diseño navegacional.

Por último, destacar que OO-H prevé la extensión del modelo de referencia con nuevas estructuras de información y/o algoritmos de análisis, según se vayan detectando nuevas necesidades en las distintas aplicaciones implementadas. En este sentido, la capacidad de OO-H de interconexión con módulos preexistentes porporciona un marco ideal para la interacción futura con módulos que incluyan algoritmos y técnicas de personalización avanzados provenientes de otras áreas de conocimiento.

Volver al Índice/Tornar a l'índex

8. El modelo lógico de presentación en OO-H

La experiencia pasada, si no se olvida, es una guía para el futuro.

PROVERBIO CHINO

Todos los conceptos presentados hasta este capítulo describen, tal y como el lector habrá podido apreciar, el espacio del problema (análisis y diseño) que pretendemos modelar. Sin embargo, para lograr la producción de interfaces web operativas, es necesario proporcionar mecanismos que faciliten una transición natural entre este espacio del problema y el espacio de la solución. En OO-H esta transición se produce de manera prácticamente transparente al diseñador, mediante la transformación automática del DAN (presentado a lo largo de los capítulos 5 y 6) en un conjunto de *páginas abstractas* (a partir de ahora páginas), entendidas como instancias de plantilla que, tal y como comentamos en el capítulo 3, representan una unidad lógica de información, es decir, un conjunto de átomos de información (posiblemente relacionados) que se visualizan en un solo paso de navegación con el fin de facilitar su comprensión. Estas páginas están interrelacionadas mediante un conjunto de enlaces, cuya característica principal es que siempre implican un salto navegacional por parte del usuario[1]. De este modo se aproxima la especificación de la interfaz a un modelo de implementación web basado en recursos (ficheros), lo cual a su vez permite:

[1] Algunos de estos saltos de navegación pueden quedar implícitos (ocultos) en la interfaz final si, a nivel de diseño de presentación, se definen distintas zonas de presentación que permitan la visualización simultánea de varias páginas abstractas en una sola página física, tal y como veremos en la sección 8.4.

- Refinar la estructura de navegación de la interfaz a un nivel de abstracción inferior (el definido por dicha página).
- Acercar la descripción de la aplicación a un tipo de unidad reconocible por el diseñador de apariencia (quien, recordemos, no tiene por qué coincidir con el que diseña el contenido y/o la navegación de dicha interfaz) y sobre el que resulta muy intuitivo proceder al diseño de la presentación.
- Prototipar aplicaciones dentro del propio entorno de desarrollo de OO-H[2].

En este capítulo comenzaremos justificando la utilidad de proporcionar una descripción textual, basada en una taxonomía de plantillas, de la interfaz objeto de diseño (ver sección 8.1). Del conjunto de instancias derivadas a partir de esta taxonomía de plantillas, aquéllas que corresponden a las distintas páginas de contenido y sus interrelaciones conforman un nuevo tipo de diagrama, denominado *Diagrama de Presentación Abstracta* (DPA), que será presentado en la sección 8.2. Una primera versión de este diagrama puede ser generada directamente a partir de la información contenida en el DAN (que fue presentado en los capítulos 5 y 6), tal y como veremos en la sección 8.2.1. A partir de este momento, la manipulación y refinamiento de cada página se puede realizar a dos niveles:

- A nivel del propio DPA. En este nivel OO-H permite tanto la inclusión de nuevas páginas estáticas como la aplicación, de forma automática, de cualquier tipo de refinamiento contenido en un Catálogo de Patrones de Interfaz, que presentaremos en la sección 8.3.
- A nivel de página individual. En este nivel, un conjunto de *Diagramas de Diseño Visual* (DDV), uno por cada página contenida en el DPA, permite la manipulación de la apariencia y contenido estático de la interfaz: composición de páginas abstractas en una sola página física, inclusión de imágenes, adición

[2] Este concepto será detallado en el capítulo 9.

de etiquetas, etc. La apariencia y semántica del DDV será presentada en la sección 8.4.

Por último, las conclusiones del capítulo serán presentadas en la sección 8.5.

8.1 La especificación de plantillas de OO-H

El uso de plantillas en las aproximaciones hipermediales no es una idea novedosa; aproximaciones como (Schwabe *et al.*, 1996; Atzeni *et al.*, 1998; Fernández *et al.*, 1998; Fraternali & Paolini, 1998; Nanard *et al.*, 1998; Gaedke *et al.*, 1999; Mecca *et al.*, 1999a; Kraus & Koch, 2002) también utilizan este mecanismo, y para ello todas ellas (hasta lo que alcanza nuestro conocimiento) se basan hoy en día en el estándar XML (W3C, 2000). OO-H (Cachero *et al.*, 2000c; Gómez *et al.*, 2001a) coincide con todas estas aproximaciones ya que, como ya comentamos en el capítulo 3, el disponer de una descripción textual basada en XML de los distintos modelos creados en el seno de las distintas aproximaciones permite la representación de su semántica mediante un lenguaje independiente de plataforma que además es extensible, no ambigüo, fácil de validar contra una sintaxis predefinida, fácil de generar y fácil de leer.

Dentro de esta especificación textual, existen dos tendencias claramente diferenciadas. Mientras que algunas aproximaciones dividen la información en documentos en función de determinados criterios (e.g. en función de tareas (Atzeni & Parente, 2001)), otras por el contrario proporcionan un solo documento que contiene toda la descripción del sistema (e.g. WebML (Ceri *et al.*, 2000)). Desde OO-H pensamos que la separación de los distintos aspectos que conforman la interfaz proporciona una serie de ventajas, entre las que destaca una localización exacta del tipo de información deseada y, como consecuencia de ésta, la restricción del ámbito al que afectan los cambios, minimizando el riesgo de que éstos afecten inadvertidamente a otros elementos de la especi-

ficación. Es por ello que OO-H ha optado por la definición de una taxonomía extensible, que en este momento incluye seis tipos de plantillas que separan explícitamente el contenido, navegación y presentación de la aplicación. Dentro de esta presentación, se ha realizado una subdivisión entre aspectos relativos a la apariencia, ubicación de elementos, composición e inclusión de elementos externos. A continuación veremos en más detalle cada uno de ellos.

8.1.1 Tipos de Plantillas en OO-H

Tal y como hemos comentado en la sección anterior, OO-H define cada una de sus plantillas en base una especificación XML, compuesta por un conjunto de DTD's[3]. Cada DTD especifica el conjunto de reglas que debe cumplir un documento XML para ser considerado un documento válido de ese tipo, e incluye las etiquetas que puede contener, las anidaciones válidas de etiquetas, los atributos y los valores permitidos. Además, cada DTD también contiene otro tipo de información, como son instrucciones de proceso, comentarios, declaración de tipo de documento, etc.

Los seis DTD's definidos en OO-H son[4]:

- *tForm* (plantilla de contenido): plantilla que permite la generación a partir del DAN de un conjunto de instancias denominadas en OO-H *páginas abstractas*. Esta plantilla proporciona por tanto las etiquetas necesarias para definir y estructurar los átomos de información que componen cada una de estas páginas abstractas. Sus elementos principales son dos: agrupaciones de elementos (cada uno de los cuales puede ser a su vez una nueva agrupación, un formulario o directamente un atributo de dominio) y formularios de interacción con el usuario. Cada formulario incluye la descripción de su acción asociada, el conjunto de campos que representa cada uno de los parámetros que deben ser enviados con la acción (incluido el modo de introducción de los

[3] *Document Type Definition* (definición de tipo de documento).

[4] Los lectores interesados en la composición exacta de cualquiera de estas plantillas pueden consultar el Apéndice A.

parámetros que comentábamos en el capítulo 5), un elemento de invocación de la acción y, opcionalmente, un elemento de reinicio del formulario.

- *tLink* (plantilla de enlazado de páginas): al igual que *tForm*, esta plantilla se instancia con la información contenida en el DAN, y permite definir el modo en que se relacionan las instancias de *tForm* generadas como parte de la interfaz. Esta especificación va más allá de la determinación del origen y el destino de cada enlace, e incluye información acerca de patrones de navegación asociados, paginación del objeto destino, etc. Es precisamente esta separación explícita de la navegación (en un documento independiente) lo que convierte la definición de la interfaz en OO-H en un Sistema Hipermedial Abierto[5] (Casteleyn & Troyer, 2002).

- *tStyle* (plantilla de estilos): esta plantilla define los rasgos relativos a colores, tamaños, tipos de letra, bordes, etc. de cada elemento de la interfaz, en base a un conjunto de reglas de visualización. Además, mediante esta plantilla se puede especificar la importación de estilos predefinidos[6].

- *tReference* (plantilla de referencias): esta plantilla se estructura en base a las páginas abstractas definidas en la interfaz (documentos de tipo *tForm*), y tiene dos funciones principales. Por un lado contiene referencias a los elementos incluidos en el *tForm* y el *tLink*. Estas referencias se distinguen por el atributo *external=false*, y su inclusión en tReference permite asociarles estilos (reglas de visualización, definidas en *tStyle*). Por otro lado este fichero también contiene referencias a elementos externos (*external=true*), como pueden ser imágenes, *applets*, etiquetas de texto arbitrarias, etc.

  En ambos casos, el atributo *tipo* asociado a los elementos de *tReference* es el encargado de especificar la clase de elemento con la que estamos tratando.

[5] *Open Hypermedia System*

[6] El uso de plantillas de estilos es muy común en las aplicaciones hipermediales actuales, estén respaldadas o no por un método de desarrollo hipermedial subyacente.

- *tLocation* (plantilla de ubicación de elementos): La plantilla *tLocation*, también organizada en base a las distintas páginas abstractas generadas a partir de *TForm*, contiene referencias a elementos de documentos *tForm* (átomos de información, formularios de interacción), *tLink* (enlaces) y *tReference* (elementos externos), y define la ubicación de cada uno de estos elementos en coordenadas relativas al área de pantalla en que se ubique dicha página abstracta.
- *tLayout* (plantilla de composición de páginas): Por último, el documento instanciado a partir de la plantilla *tLayout* define las posibles distribuciones de páginas abstractas en páginas físicas[7]. Cada posible distribución se almacena en un elemento de tipo *tLayoutGroup*. Dentro de este elemento, las coordenadas y tamaño de cada zona de pantalla que lo componen se define mediante elementos de tipo *tLayoutElement*. Cada uno de estos elementos contiene referencias a las páginas abstractas (instancias de *tForm*) que se deben visualizar en él.

De esta especificación se puede inferir que la descripción textual del sistema está compuesta por un conjunto de páginas abstractas, obtenidas como instanciación de la plantilla *TForm*, que se ven complementadas con cinco documentos (una instancia de tLink, una de tReference, una de tStyle, una de tLocation y una de tLayout) que actúan como decoradores (Gamma *et al.*, 1995) de dichas páginas abstractas.

Es importante destacar cómo en OO-H la inclusión de esta especificación textual ha permitido diseñar los compiladores de modelos (que veremos en el capítulo 9) a partir de ella, independizándolos de este modo del modelo de referencia de OO-H. Así, cualquier método capaz de generar una descripción XML compatible con los DTD's de OO-H puede ser utilizado como base sobre la refinar la interfaz (ya en el seno del entorno de desarrollo de OO-H) y aplicar los generadores, del mismo modo que se haría con una

[7] Esta distribución se materializará durante la fase de generación en distintos marcos de página o cualquier otro tipo de mecanismo similar.

especificación obtenida mediante el método presentado en este trabajo.

8.2 El Diagrama de Presentación Abstracta

De los tipos de plantilla presentados en la sección anterior, ya hemos comentado cómo las instancias de plantilla *TForm* y *tLink* son las encargadas de definir un esqueleto de implementación web que OO-H explicita mediante un nuevo diagrama, el *Diagrama de Presentación Abstracta* (DPA). Este diagrama refleja de una manera visual cada instancia de página abstracta, así como sus interconexiones con el resto de páginas abstractas del diagrama. La configuración inicial de esta estructura se extrae directamente del DAN, a partir del atributo *ubicación del efecto*[8] (en origen o en destino) y del patrón navegacional asociado a los enlaces navegacionales. Asimismo, el grueso del contenido de cada página abstracta se obtiene de la especificación de clases, atributos y servicios navegacionales incluidos en el DAN. A continuación veremos el proceso de generación de este diagrama por defecto.

8.2.1 Generación del DPA por defecto

La construcción del DPA comienza con una fase de generación automática de su estructura por defecto. Para ello, los patrones capturados a nivel de DAN, junto con las características de objetos y atributos, proporcionan la información mínima necesaria para generar de forma automática dicho diagrama[9].

Para realizar este proceso, OO-H define un conjunto de reglas de traducción entre los distintos elementos del DAN y su especificación equivalente en el DPA. Dichas reglas son:

[8] Este concepto fue presentado en la sección 5.3.5.

[9] Tal y como comentaremos en el capítulo 9, esta estructura debe regenerarse siempre que se modifique dicho DAN, con el fin de actualizar los documentos de tipo TForm y TLink que recogen la información que puede haberse visto afectada por dichos cambios.

- V-Atributos y parámetros de formularios (excepto los ocultos) contenidos entre dos enlaces de navegación en destino: se convierten en componentes de un nuevo documento de tipo *tForm*, anidados convenientemente en función de las características de los enlaces de navegación con *ubicación del efecto=origen* que relacionan los distintos elementos. Además, el modo de introducción asociado a cada uno de estos parámetros puede dar lugar a una subestructura de páginas de complejidad arbitraria. Por último, si el método subyacente devuelve algún tipo de resultado, se debe generar una nueva página abstracta que permita mostrar esos resultados. Todas estas páginas se interrelacionan mediante un conjunto de componentes que se insertan en el documento de tipo *tLink*.
- R-Atributos: provocan la aparición en el diagrama de una nueva página abstracta de tipo *tForm*, que contiene todos los atributos referenciados junto con los atributos que el diseñador determine que identifican al objeto original. La página resultante actúa de este modo como una especie de vista extendida sobre el objeto. Además, los R-Atributos causan la aparición de un nuevo elemento de la plantilla *tLink* que enlaza de manera bidireccional esta nueva página desde la original.
- Enlaces de navegación con *ubicación del efecto=destino*: se transforman en elementos de tipo enlace en la plantilla de tipo *tLink* asociada a la aplicación.
- Colecciones cuya entrada es un enlace de navegación en destino: se transforman en una nueva página de tipo *tForm*. Este documento incluirá toda la información de los enlaces que, partiendo de la colección, tengan una *ubicación del efecto=origen*. Además, se generará un nuevo elemento de *tLink* por cada enlace de la colección con *ubicación del efecto=destino*.
- Patrones de navegación de tipo índice: generan una nueva página de tipo *tForm*, que contiene los elementos que conforman el índice. Además, se generará un nuevo elemento en la plantilla *tLink* que materialice dicho índice.

- Patrones de navegación de tipo visita guiada: generan un conjunto de elementos en la plantilla *tLink* que materializa la herramienta de navegación *siguiente, anterior, primero y último* asociada a las colecciones de objetos afectadas por este tipo de patrón.
- Filtros de navegación que incluyen valores que deben ser introducidos por el usuario: generan una nueva página de tipo *tForm*, que contiene los elementos de consulta del filtro, y un enlace en el documento de tipo *tLink* que une esta página abstracta con la página destino del enlace navegacional al que se hallaba asociado el filtro.

Ya en el entorno de desarrollo de OO-H, un algoritmo que tiene en cuenta estas reglas es capaz de generar una estructura de páginas abstractas y sus interrelaciones, tal y como veremos a continuación.

Aplicación al caso de estudio. Con el fin de ilustrar este proceso de generación de páginas abstractas, en la Fig. 8.2 presentamos una vista parcial del DPA correspondiente a la interfaz que da respuesta a los requisitos *Ver Ocupación Hotel* y *Realizar Nueva Reserva*, cuyo DAN (presentado en el capítulo 5) reproducimos en la Fig. 8.1.

En este ejemplo podemos ver cómo dicho DAN ha dado lugar a una estructura de páginas de tipo *tForm* delimitadas por un conjunto de enlaces que se almacenan en una página de tipo *tLink*, común a toda la interfaz.

En este diagrama, el R-Enlace *Subsistema de Reservas* (ver Fig. 8.1) genera un enlace de DPA homónimo que da entrada a una página *Reservas*. Esta página no incluye ningún elemento de información, ya que sólo contiene un menú de acceso (materializado mediante dos elementos en *tLink*) a dos nuevas páginas: una página *Cliente* que muestra la ocupación del hotel, y que recoge información del cliente, la habitación y los datos de reserva, y una página *Habitación* que muestra un listado de habitaciones, y

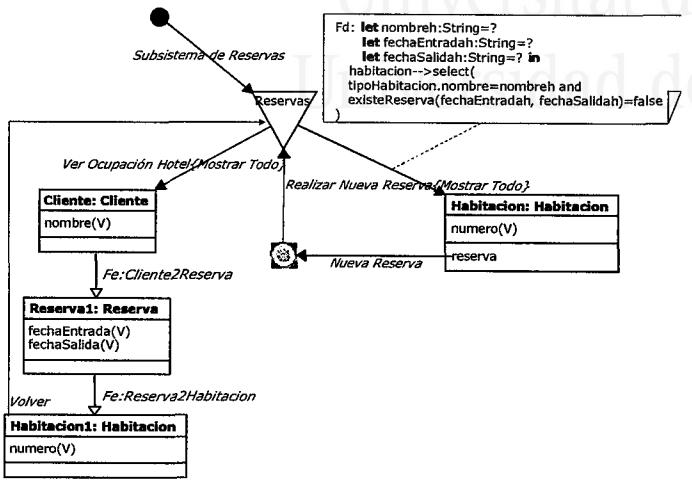

Figura 8.1. DAN de Segundo Nivel refinado correspondiente a los requisitos *Ver Ocupación Hotel* y *Realizar Nueva Reserva*

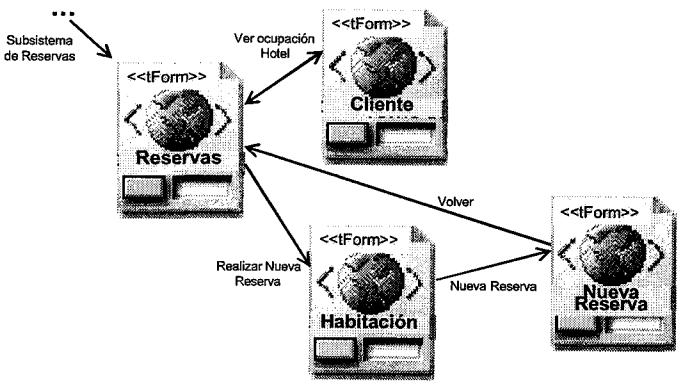

Figura 8.2. Vista parcial del DPA generado a partir del DAN correspondiente a los requisitos *Ver Ocupación Hotel* y *Realizar Nueva Reserva*

desde la cual es posible invocar al método *Realizar nueva reserva* mediante el formulario contenido en la página *Nueva Reserva.*

Nuevamente, el *ámbito de aplicación=destino* del S-Enlace *Nueva Reserva* (ver Fig. 8.1) causa la generación de una nueva página de tipo *tForm* que gestiona el modo en que se recogen los parámetros

de dicho servicio. La interconexión y el número de páginas que intervienen en esta recogida de parámetros depende del modo de introducción de dichos parámetros. En este ejemplo, por simplicidad, supondremos que todos son de tipo *inmediato*, y por tanto se piden a la vez en el contexto de un formulario (motivo por el que en el DPA aparece una sola página abstracta).

Una vez invocado el método y devuelto el control a la interfaz, y debido a que la especificación del servicio no incluye ningún parámetro de retorno, dicha interfaz vuelve a mostrar la página del menú de opciones, tal y como podemos ver en la Fig. 8.1. A nivel de DPA, esto causa la generación de un nuevo elemento en la instancia de *tLink*, que corresponde al enlace que une la página de recogida de parámetros con la página de menú.

Como ejemplo de la estructura interna de cada una de estas páginas, el documento XML correspondiente a la página de *Ocupación* (generado a partir del DTD presentado en el Apéndice A) puede ser visto en la Fig. 8.3.

En este documento, observamos cómo el elemento superior en la jerarquía de anidación es la clase navegacional cliente, que es la colección de objetos de entrada del enlace *Ver ocupación hotel* (cuyo identificador interno es *LI3*). A partir de ahí, se han ido anidando objetos siguiendo los enlaces en origen de la Fig. 8.1.

El DPA por defecto proporciona una nueva vista sobre la interfaz que facilita la comprensión y el acceso a los distintos documentos que componen su descripción textual. Sin embargo, y a pesar de que a partir de esta descripción ya se puede materializar una interfaz que dé respuesta a todos los requisitos funcionales, la especificación generada a partir del DAN a menudo no está lo suficientemente refinada, por lo que aún no se puede hablar de producto final. Es por ello que, para conseguir dotar a la aplicación de una interfaz más sofisticada, el diseñador puede realizar una serie de refinamientos sobre este diagrama, refinamientos que se verán reflejados en la interfaz finalmente generada y que pasamos a detallar a continuación.

```
<?xml version="1.0" encoding="UTF-16"?>
<!-- Localizacion del DTD externo asociado al documento: -->
<!DOCTYPE tForm SYSTEM "C:\Archivos de programa\LES Objects\XMLLib\tForm.dtd">
<!-- Codigo XML de la pagina abstracta: -->
<tForm id="tForm_Cliente" idGeneratorLink="LI3">
    <collection id="collection_Cliente" idGeneratorLink="LI3" query_associed="true">
        <!-- Estructura de datos proporcionada por la clase Cliente: -->
        <object class="Cliente" id="Cliente_object">
            <!-- Datos relativos a los atributos visibles de la clase Cliente: -->
            <attrib id="Cliente.nombre" type="number"/>
            <!-- Datos relativos a clases navegables (ShowIn=Origen) desde Cliente
            con al menos un rol activo entre ambas: -->
            <collection id="collection_Cliente.Reserva1" idGeneratorLink="LI5">
                <!-- Estructura de datos proporcionada por la clase Reserva1: -->
                <object class="Reserva1" id="Cliente.Reserva1_object">
                    <!-- Datos relativos a los atributos visibles de la clase Reserva1: -->
                    <attrib id="Cliente.Reserva1.fechaEntrada" type="number"/>
                    <attrib id="Cliente.Reserva1.fechaSalida" type="number"/>
                    <!-- Datos relativos a clases navegables (ShowIn=Origen) desde Reserva1 con al menos
                    un rol activo entre ambas: -->
                    <collection id="collection_Cliente.Reserva1.Habitacion1" idGeneratorLink="LI6">
                        <!-- Estructura de datos proporcionada por la clase Habitacion1: -->
                        <object class="Habitacion1" id="Cliente.Reserva1.Habitacion1_object">
                            <!-- Datos relativos a los atributos visibles de la clase Habitacion1: -->
                            <attrib id="Cliente.Reserva1.Habitacion1.numero" type="number"/>
                        </object>
                    </collection>
                </object>
            </collection>
        </object>
    </collection>
</tForm>
```

Figura 8.3. Documento XML asociado a la página abstracta *Ocupación*

8.2.2 Refinamientos del DPA

Una vez generado el DPA por defecto, el proceso de refinamiento de la interfaz puede implicar la modificación tanto de su estructura como del contenido de las distintas plantillas. Para soportarlo, OO-H prevé tres tipos de actividad:

- Adición de páginas estáticas directamente sobre el DPA.
- Aplicación de un lenguaje de patrones, contenido en un Catálogo de Patrones de Interfaz.
- Modificación de las páginas de estructura individuales mediante la manipulación visual del Diagrama de Diseño Visual (DDV) que representa la configuración de cada página de la interfaz.

En primer lugar, la adición de páginas estáticas y su conexión con el resto de páginas abstractas de la interfaz se realiza directamente sobre el DPA. Este proceso viene auspiciado por la necesidad de

integración en determinados puntos de la interfaz del trabajo de diseñadores gráficos (e.g. zonas promocionales[10]), muy comunes en las aplicaciones web tradicionales.

Sin embargo, muchos otros tipos de refinamiento afectan a elementos ocultos a nivel de DPA, y que deben ser modificados por tanto directamente sobre la especificación textual subyacente. OO-H no recomienda realizar este tipo de cambios de manera manual, y para evitarlos proporciona un *Catálogo de Patrones* que, aplicados sobre el DPA, pueden modificar su estructura tanto externa (páginas y sus relaciones) como interna (contenido de las páginas XML que definen la interfaz). Este catálogo será tratado en más profundidad en la sección 8.3. Este catálogo no sólo permite modificar los modelos de manera automática, sino que agiliza el refinamiento de las aplicaciones al mismo tiempo que proporciona un catálogo de prácticas efectivas para el incremento de facilidad de uso de la interfaz.

Por último, para la modificación de la apariencia visual de la interfaz (composición de páginas abstractas para formar páginas físicas, disposición y estilos de los distintos elementos que componen la interfaz, tipo de elementos de interacción con el usuario para recoger la información requerida, etc.) se ha optado por un entorno visual, materializado en un conjunto de Diagramas de Diseño Visual (DDV) que permiten la edición de todos los elementos relativos a la ubicación y diseño gráfico de la interfaz. Este diagrama será presentado en la sección 8.4.

8.3 Uso de Patrones Hipermediales en OO-H

Un patrón puede ser definido como *una solución o conjunto de soluciones ante un determinado problema que, habiendo sido aplicadas de forma recurrente en aplicaciones existentes (y por tanto convenientemente testadas), han sido recogidas de manera sistemática para facilitar su difusión y reuso.* Del mismo modo, un

[10] Un ejemplo clásico en este tipo de zonas es el diseño de túneles de entrada/salida a la aplicación.

patrón hipermedial puede definirse como *un conjunto de técnicas de probada eficacia para la resolución de un determinado problema en el ámbito de las aplicaciones hipermediales, junto con un conjunto de reglas para combinarlos en un estilo de arquitectura (en este caso, el propiciado por dichas aplicaciones hipermediales).*

Las decisiones del diseñador para solucionar estos problemas se pueden detectar a varios niveles: desde el nivel de arquitectura (e.g. cómo estructurar los módulos de la aplicación y qué responsabilidades asignar a cada uno de ellos) hasta el nivel de implementación (e.g. cómo presentar al usuario una selección entre un grupo de objetos). Es por ello que OO-H distingue distintas categorías de patrones[11], cada una de las cuales afecta a un subconjunto de rasgos de la aplicación. Estas categorías son:

- Patrones de Arquitectura. Para definirlos, OO-H ha adoptado la propuesta de Conallen (Conallen, 1999) y distingue entre tres tipos de arquitecturas de aplicación web: (1) Cliente web ligero, (2) Cliente web pesado y (3) Aplicación web distribuida. En el primer caso nos encontramos ante un cliente que actúa como mero navegador de la información, sin capacidades de cómputo adicional. En el segundo caso (cliente pesado) se presupone que el cliente tiene cierta capacidad de proceso, aunque existen reglas estrictas que gobiernan dicha capacidad. En este caso el diseñador puede delegar en el cliente ciertas responsabilidades, como puede ser la validación de campos, gestión de asistentes, elementos de dibujado de gráficas de datos, etc. Para ello el diseñador puede decidir generar y/o enviar fragmentos de código (*applets, controles ActiveX, etc.* en un lenguaje arbitrario que permitan extender los navegadores con dicha funcionalidad. Por último en las aplicaciones web distribuidas se definen componentes cliente y servidor, y se establece entre ellos unos mecanismos de comunicación, que no tienen por qué estar basados en HTTP.

[11] Esto implica que, con el término genérico de patrón, OO-H engloba desde marcos de dominio (*frameworks*) a patrones de implementación (elección del constructor de implementación (*widget*) que mejor se adapta a las necesidades del usuario).

- Patrones de estructura (Schwabe *et al.*, 2001). Los patrones de estructura se definen como un diseño reutilizable construido a partir de un conjunto de clases abstractas y/o concretas y un modelo de colaboraciones entre objetos (Schwabe *et al.*, 2001). Este tipo de patrones proporciona una plantilla extensible para aplicaciones dentro de un dominio específico. Por ejemplo, OO-H proporciona un marco de clases para trabajar con aplicaciones de comercio electrónico que implementan los conceptos *producto*, *cliente*, *carrito de la compra*, *factura* y *línea de factura*, así como las relaciones entre ellas. Otro ejemplo es el marco de personalización de OO-H que recoge un modelo de usuario básico, y que presentamos en el capítulo 7.

  El ámbito de aplicación de este tipo de patrones en OO-H es el modelo de dominio de la aplicación, y de hecho el entorno de desarrollo de OO-H los guarda como instancias incompletas de diagramas de clases. El uso de estos marcos en OO-H se realiza seleccionando el marco adecuado, que se encuentra catalogado en función del dominio al que pertenece. Esta selección causa, de manera automática, la generación de las estructuras correspondientes en el diagrama de clases, a partir de las cuales el diseñador puede seguir trabajando para adaptarlas al problema concreto de que se trate.

- Patrones de diseño de interfaz. Los patrones de diseño de interfaz son patrones que afectan al modo en que el usuario navega e interacciona con la aplicación. Existen multitud de patrones definidos por la comunidad científica que pueden ser catalogados dentro de este ámbito (Rossi *et al.*, 1997; Nanard *et al.*, 1998; Rossi *et al.*, 1999; Nanard & Nanard, 1999; Paolini & Garzotto, 1999; Germán & Cowan, 2000). En OO-H hemos capturado un pequeño exponente de dichos patrones, cuyo número esperamos ir ampliando en un futuro. La aplicación de este tipo de patrones en OO-H puede afectar tanto al DAN como al DPA y/o a la especificación textual subyacente.

- Patrones de implementación de interfaz. Este tipo de patrones permiten determinar el constructor de implementación que mejor se adapta a las necesidades y características de la audiencia

objetivo de la aplicación. OO-H actualmente no tiene definido ningún patrón de esta categoría, por lo que aplica de manera automática un conjunto de constructores por defecto.

En este trabajo nos hemos centrado en la definición de patrones de diseño de interfaz y, dentro de ellos, en aquéllos relacionados con la semántica de la navegación e interacción, que son los que tienen más relevancia desde el punto de vista del DPA[12]. Estos patrones se almacenan como parte de un Catálogo de Patrones de Interfaz, tal y como veremos a continuación.

8.3.1 El Lenguaje de patrones hipermediales de OO-H

Como ya hemos comentado anteriormente, la complejidad de una modificación manual de determinados rasgos de la interfaz hace inviable su aplicación. Es por ello que OO-H ha optado por definir un *lenguaje de patrones hipermediales*, entendido como *una colección parcialmente ordenada de guías, en un formato estándar, para dar solución a problemas recurrentes en el ámbito del diseño de interfaces hipermediales* (Rossi *et al.*, 1997).

En general, las aproximaciones basadas en lenguajes de patrones tienen como ventaja principal una definición de proceso controlada sin resultar restrictiva; en una aproximación de estas características todas las decisiones tomadas durante el proceso de diseño quedan automáticamente documentadas (por la propia especificación del patrón) y disponibles para su consulta por parte de los programadores. Una vez que se define un lenguaje de patrones para un contexto dado, los desarrolladores noveles que aprendan dicho lenguaje pueden utilizarlo como una herramienta para discutir sobre decisiones y posibilidades de diseño, ya que proporciona un vocabulario común rápidamente comprendido por otros desarrolladores familiarizados con el uso de este mecanismo de abstracción. Al mismo tiempo, los lenguajes de patrones muchas

[12] La extensión del trabajo para incluir patrones relacionados con las metáforas de presentación de la interfaz queda como trabajo futuro.

veces aportan soluciones para problemas con los que el desarrollador no está familiarizado, minimizando de esta forma la tendencia a reinventar soluciones, normalmente de menor calidad y con un coste superior. Otra ventaja de este tipo de lenguajes se observa a la hora de mantener el sistema: los patrones pueden ser usados para constreñir los cambios permitidos sobre el diseño del sistema, así como para asegurar que dichos cambios se realizan de manera correcta, sobre todo si, como en el caso de OO-H, es posible proceder a la automatización de su aplicación.

En OO-H este lenguaje de patrones se almacena en un Catálogo de Patrones, cuyo propósito es triple:

- Recoger experiencias de diseño que han sido probadas de manera exhaustiva, y se han comprobado efectivas para la resolución de un determinado problema.
- Promover la aplicación automática de esos patrones sobre los modelos OO-H, con el fin de minimizar el tiempo de desarrollo e incrementar la calidad de los sistemas resultantes.
- Homogeneizar las características de la interfaz de la aplicación.

El primer objetivo (documentar experiencias de diseño efectivas) está en la base de la propia definición de los patrones, y por tanto una correcta formulación de los mismos garantiza su consecución.

Por otro lado, en el capítulo 5 comentamos cómo era posible aplicar patrones de navegación sobre los enlaces navegacionales en el DAN con una sola pulsación de ratón, y cómo la aplicación de este tipo de patrón automáticamente determinaba el número, enlazado y contenido de las páginas abstractas generadas en el DPA. A nivel de DPA la automatización de este tipo de patrones sigue un proceso distinto, y se realiza mediante la definición, para cada uno de ellos, de un conjunto de *Reglas de Transformación*, una para cada implementación posible del patrón.

Una regla de transformación es un procedimiento Python[13] (Python, 2002) que es capaz de actuar sobre el repositorio y/o sobre la

[13] Las ventajas de uso de este lenguaje serán expuestas en la sección 9.2.1.

especificación textual (documentos XML) de la interfaz (dependiendo del nivel al que se aplique) y modificar los rasgos que sean necesarios. Estas reglas de transformación pueden ser aplicadas a diferentes niveles: desde un nivel de esquema (DAN o DPA) a un nivel de página abstracta. Las reglas de transformación dirigen por tanto los cambios en el diagrama donde se aplique el patrón. Estos cambios pueden ser de cualquier tipo: creación de nuevas páginas, cambios en el mapa de navegación entre páginas, cambios en características de la presentación, etc. y por tanto pueden modificar tanto la estructura externa de los diagramas (características visibles de los constructores a nivel de DAN o páginas y sus relaciones a nivel de DPA) como interna (características de esos constructores, páginas y enlaces, estilos, ubicación de páginas, etc.).

Por último, la posibilidad de ofrecen las reglas de transformación a la hora de aplicar los distintos patrones a nivel de diagrama en su conjunto incide de manera positiva sobre el tercer objetivo: garantizar la homogeneidad de determinados rasgos (aquéllos determinados por el patrón) de la interfaz resultante.

Como ejemplo supongamos que queremos aplicar un patrón denominado *patrón de anuncio de destino*[14], que causa que todos los enlaces de la interfaz dispongan de manera automática de un texto que especifica, en todos los casos, la información, servicios y opciones de navegación disponibles si el usuario se decide a navegar por ese enlace.

La modificación manual de la interfaz para incluir esta característica supondría tener que editar distintos documentos XML afectados por el patrón. En primer lugar habría que editar la instancia de plantilla *TReference* para añadir una etiqueta de texto por cada cada enlace, cuyo contenido (contenido de la página destino, posibilidades de navegación y servicios disponibles) tendría además que ser introducido por el propio diseñador. Además, habría que editar la instancia de plantilla tLocation para ubicar dichas etiquetas al lado de cada elemento de tipo enlace (o,

[14] La enumeración y catalogación de patrones en OO-H será presentada en la sección 8.3

alternativamente, como texto que apareciese al pasar el ratón por encima).

La existencia de una regla de transformación como la proporcionada por OO-H automatiza todo el proceso, y permite aplicar el patrón con la implementación deseada (en este caso texto visible al lado del enlace u oculto mientras el usuario no pase el ratón por encima del mismo) con una simple opción de menú. Además, utilizando esta regla de transformación nos aseguramos de que todos los enlaces van a disponer de esta característica, y que el texto asociado a cada uno de ellos va a proporcionar la información deseada, lo cual no sólo garantiza la homogeneidad del tratamiento de esta característica, sino que simplifica la tarea de testeo final de la aplicación.

8.3.2 La especificación de patrones en OO-H

Para la especificación de patrones, OO-H ha optado por el estilo definido en (Buschmann *et al.*, 1996), que a su vez se basa en la especificación presentada por (Alexander, 1979). El motivo ha sido su especial adecuación para patrones abstractos con varios posibles modos de implementación, que son dos de las características principales de los patrones hipermediales. Este formato ha sido además enriquecido con cinco nuevas secciones que permiten su aplicación automática como parte del método y que son:

- *Ámbito de aplicación*: determina sobre qué diagrama y/o sobre qué constructores de OO-H se pueden ejecutar estos patrones.
- *Conjunto de Reglas de Transformación*: una por cada posible implementación del patrón.
- *Implementación por defecto*: cada patrón debe definir una de sus posibles implementaciones como *por defecto*, con el fin de ayudar al diseñador a obtener los rasgos de interfaz deseados con un esfuerzo mínimo.
- *Ejemplos de Uso*: de manera opcional, y con el fin de facilitar el acceso y el uso al diseñador, cada patrón puede incluir una

sección de tipo *Ejemplos de Uso* donde se recogen lugares web donde dicho patrón ha sido aplicado con éxito.

- *Restricciones de aplicación*: características técnicas, legales o de cualquier otra índole que pueden influir en la conveniencia de aplicar un patrón, ya sea en general o referido a alguna de sus variantes de implementación.

Un ejemplo de Patrón. Con el fin de ilustrar las extensiones que OO-H ha realizado sobre la definición de los distintos patrones, vamos a suponer que deseamos aplicar un nuevo patrón, denominado *patrón de ayuda*, sobre el DPA de la Fig. 8.2. Este patrón asocia una descripción adicional a cada página abstracta de la interfaz.

Una definición simplificada de este patrón, en la que pueden verse resaltadas las secciones incluidas por OO-H, sería la siguiente:

- Nombre: Patrón de Ayuda.
- **Nivel de aplicación**: DPA
- Contexto: Un componente proporciona información acerca del significado y utilidad de la vista de usuario actual.
- Problema: La falta de familiaridad de un usuario con el dominio de la aplicación y/o con los mecanismos de uso de las interfaces hipermediales pueden causar un bloqueo durante el uso de la misma. Para evitar este problema e incrementar la facilidad de uso de la interfaz, se necesita un mecanismo que proporcione información adicional acerca del propósito y significado del contenido de cada página abstracta. Dos fuerzas están asociadas con este problema:
  - No se conoce el problema exacto que está causando el bloqueo en el usuario.
  - El componente de ayuda debería estar débilmente acoplado: la vista actual del sistema no debería depender de detalles de los componentes de ayuda.
- Solución: Implementar un mecanismo en el cual un cambio de vista cause un cambio-propagación que reestablezca la consistencia entre la vista y la información del componente de ayuda.
- **Implementación por defecto (I1)**: Crear una nueva página abstracta de ayuda, así como un enlace en la esquina superior derecha de cada página abstracta que asocie la página actual con la página de ayuda.
  - **Regla de transformación**: ...
- Otras implementaciones:

- **I2**: Incluir un conjunto de páginas estáticas, cuyo contenido haya sido proporcionado por un diseñador de contenido, que proporcione ayuda genérica sobre la aplicación.
  - **Regla de transformación**: ...
- **I3**:Incluir un esquema gráfico de las páginas que conforman la aplicación. Este esquema se genera de manera automática a partir de la estructura del propio DPA y de los distintos DDV asociados a cada página.
  - **Regla de transformación**: ...

- **Ejemplo de uso**: Campus Virtual de la Universidad de Alicante.
- **Restricciones de aplicación**: No tiene.

Como el lector habrá podido constatar por la descripción proporcionada, en este patrón hemos proporcionado tres posibilidades alternativas de inclusión de ayuda en la aplicación: texto asociado a cada página, texto genérico y diagrama generado de manera automática a partir del propio modelo de interfaz (cada uno con su correspondiente regla de transformación)[15]. Además, podríamos decidir incluir en la interfaz final dos o incluso los tres mecanismos de manera simultánea.

En realidad, en el ámbito de las interfaces hipermediales la implementación de dos o más mecanismos para cubrir un mismo problema es un rasgo no sólo permitido sino común. Un ejemplo clásico de esta situación se da en el *Patrón de Observación de la Navegación*, que permite al usuario volver a vistas previas de la aplicación, y que suele ser capturado en la interfaz mediante la aparición simultánea del botón de *vuelta atrás* y de un *histórico* de páginas navegadas.

Además, es importante destacar que la aplicación de algunas reglas requiere que el diseñador interactúe con la herramienta para proporcionar información adicional. Es el caso por ejemplo del patrón de ayuda, que, en el caso de elegir la implementación por defecto, requiere que el diseñador introduzca dicho texto de ayuda para cada página de la aplicación.

[15] La ilegibilidad del código asociado al patrón para aquellos lectores no familiarizados con el modelo de referencia de OO-H desaconseja su inclusión en el presente documento.

Volviendo a nuestro ejemplo, el resultado de aplicar el *patrón de ayuda* sobre el DPA de la figura 8.2 puede ser visto en la Fig. 8.4.

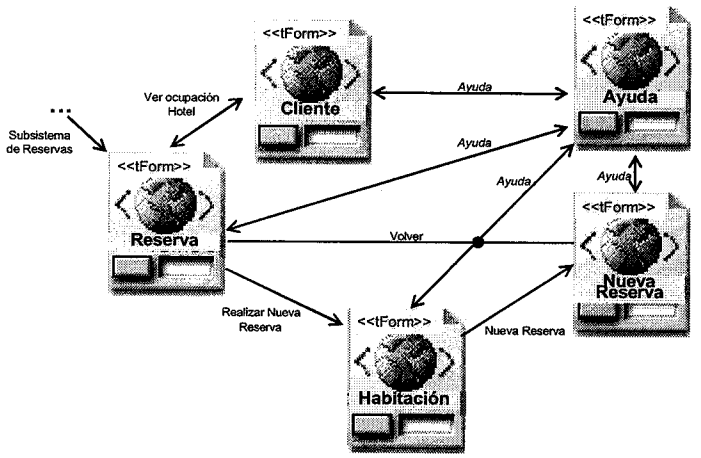

Figura 8.4. DPA tras la aplicación del *Patrón de Ayuda* por defecto

En este ejemplo vemos cómo la adición de la página de ayuda y de los enlaces correspondientes oscurece la figura, sin aportar una información demasiado relevante para el diseñador. Es por ello que OO-H permite ocultar las modificaciones causadas en el diagrama por la aplicación de un determinado patrón.

Por último, hay que tener en cuenta que no todos los patrones tienen un efecto visible en los distintos diagrama de OO-H, aunque todos causan un cambio en la estructura textual subyacente. Como ejemplo, el *Patrón de anuncio de destino* que hemos comentado anteriormente causa una modificación en los componentes de tLink y tReference, pero no añade ni elimina ninguna página ni enlace del DPA.

Para la organización de los distintos patrones de interfaz, OO-H proporciona un Catálogo de Patrones que presentamos a continuación.

8.3.3 La estructura del Catálogo de Patrones en OO-H

Tal y como puede verse en la Fig. 8.5, OO-H incluye un Catálogo de Patrones que clasifica los distintos patrones de diseño de interfaz atendiendo a su propósito[16]. Esta estructuración difiere de la presentada en otros trabajos (Germán & Cowan, 2000), debido por un lado a que en dicho catálogo OO-H obvia patrones implícitos en el propio modo de construcción de los modelos y por otro al nivel de granularidad al que se definen dichos patrones, que en OO-H se establece de manera que éstos puedan ser aplicados de manera sencilla en sus modelos.

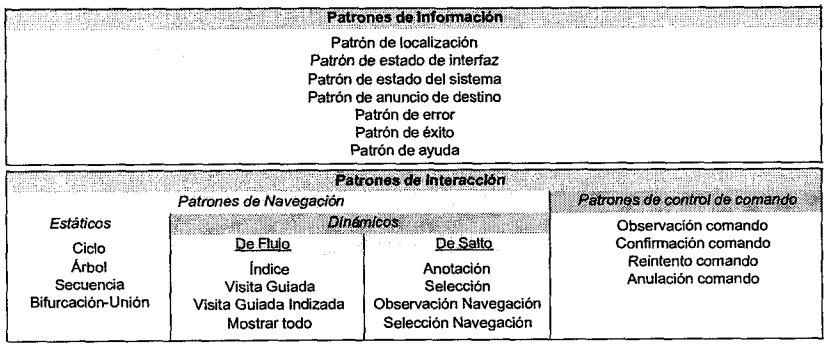

Figura 8.5. Estructura del catálogo de patrones de OO-H

Dicho catálogo se divide en dos secciones, en función del tipo de solución que aporta cada patrón:

- Patrones de información: proporcionan información complementaria acerca del contexto en que se presenta la información.

[16] Este catálogo recoge sólo un pequeño subconjunto de los patrones hipermediales propuestos por la comunidad científica. El estudio de nuevos patrones y su correspondiente implementación como extensiones a dicho catálogo quedan como trabajo futuro. En este sentido, es importante destacar cómo la arquitectura del entorno de desarrollo que presentaremos en la sección 9 ha tenido en cuenta este hecho y permite la inclusión de nuevos patrones de una manera transparente para el resto del entorno.

- Patrones de interacción: proporcionan mecanismos adicionales de comunicación entre usuario e interfaz durante las actividades de navegación e invocación de servicios.

Los patrones de información son los más conocidos, debido a los orígenes de las aplicaciones web como diseminadoras de información. Entre ellos, OO-H recoge los siguientes:

- Patrón de localización: su uso proporciona al usuario información relativa a su ubicación actual en la aplicación, contribuyendo así a evitar el *síndrome de perdido en el hiperespacio*. También se conoce como *Contexto Navegacional* (Bernstein, 1998). El nivel de abstracción al cual se aplica el patrón de localización puede variar desde un contexto de aplicación a un contexto de objeto, y para su implementación por defecto OO-H ha optado por la especificación en la parte superior de cada página abstracta del camino recorrido por el usuario (optimizado para eliminar pasos deshechos por el propio usuario) hasta llegar a la página abstracta actual.
- Patrón de información de estado de interfaz: su uso permite a la aplicación mostrar información acerca de estado en que se encuentra cada página o la interfaz en su conjunto. La implementación por defecto propuesta por OO-H es la adición de un mensaje en la barra de estado del navegador que indica si la interfaz se encuentra mostrando información, esperando entrada de usuario, esperando solicitud de envío de información al servidor o esperando recuperar el control tras una solicitud de servicio.
- Patrón de información de estado del sistema: su uso permite a la aplicación mostrar información acerca de su estado. La implementación por defecto que OO-H proporciona de este patrón consiste en la generación de ventanas *javascript* que avisan al usuario ante cambios en el entorno que afectan la visión que dicho usuario tiene de la interfaz (e.g. el cumplimiento de una fecha límite, que permite que ahora el usuario se capaz de visualizar un tipo de información que hasta entonces le estaba

vedada). Una posible implementación alternativa es la emisión de sonidos de premio o aviso.

- Patrón de anuncio de destino: su aplicación permite al usuario obtener información sobre qué objetos/servicios/opciones de navegación encontrará en caso de decidirse a atravesar un determinado enlace. La implementación por defecto que OO-H proporciona para este patrón consiste en la generación automática un campo de texto a partir de la información contenida en el DAN, que se asocia a cada enlace de la instancia de plantilla tLink.
- Patrón de error: su uso redefine el comportamiento por defecto de la interfaz ante la ocurrencia de un error, que consiste en presentar el mensaje de error tal cual ha sido generado. La aplicación de este patrón causa la generación de una nueva página abstracta donde, en caso de ser posible, se muestra un mensaje más significativo para el usuario. Para ello OO-H define sentencias de explicación para los códigos de error más comunes, y da la posibilidad de volver al lugar donde el usuario se encontraba inmediatamente antes de que se produjese el error.
- Patrón de éxito: su uso permite a la aplicación notificar al usuario la ejecución correcta de la invocación de un servicio, en caso de que dicho servicio no proporcione ningún parámetro visible que comunique ese hecho al usuario de manera directa o indirecta. La implementación por defecto de OO-H consiste en la creación de una nueva página donde se muestra un mensaje estándar que informa al usuario acerca del éxito de la operación.
- Patrón de ayuda: su uso permite al usuario tener acceso a ayuda (contextual o no) desde cualquier punto de la aplicación. La implementación por defecto de este patrón consiste en proporcionar un comando de ayuda disponible desde cada página abstracta, que se active para cada página del DPA para la que el diseñador haya proporcionado un texto de ayuda.

Por otro lado, los patrones de interacción definen el modo en que el usuario puede guiar el comportamiento de la aplicación.

Esta guía, al contrario que otras aproximaciones, no se restringe a seleccionar entre los caminos de navegación ofertados, sino que además recoge distintas posibilidades de introducir información, invocar servicios y visualizar los resultados de dicha invocación. Así, OO-H distingue entre:

- Patrones de navegación estáticos: definen estructuras de navegación entre páginas estáticas, y por tanto se aplican siempre a nivel de DPA. Entre ellos destacamos los patrones de ciclo, árbol, secuencia y bifurcación-unión.
- Patrones de navegación dinámicos. Al contrario que los estáticos, se definen sobre conjuntos indefinidos de objetos en el sistema. Dentro de estos patrones, y dependiendo de si el camino de navegación sigue un orden predefinido o no a través de los objetos, OO-H distingue entre:
  - Patrones de flujo de navegación: definen, explícita o implícitamente, caminos de navegación ordenados lineales dentro de un conjunto de objetos. Son los patrones clásicos de navegación: índice, visita guiada, visita guiada indizada y mostrar todo. Estos patrones fueron presentados en el capítulo 5, ya que su ámbito de aplicación se circunscribe al DAN. Las reglas de transformación en este caso permiten aplicar, en caso de ser necesario, el mismo tipo de patrón de flujo a todos los enlaces contenidos en cualquier subsistema, asegurando así la homogeneidad en el modo de acceso a la información.
  - Patrones de salto de navegación: definen caminos de navegación no lineales tanto dentro de un conjunto de objetos como entre diferentes conjuntos. Entre ellos destacan:
    - Patrón de anotación de enlaces, que permite definir enlaces no estructurales entre objetos y/o páginas arbitrarias del sistema.
    - Patrón de observación de la navegación (Tauscher & Greenberg, 1997; Rossi *et al.*, 1997), que permite al usuario saltar a vistas previas de la aplicación. La implementación por defecto de este tipo de patrón en OO-H consiste en la de-

finición de un mecanismo de vuelta atrás. Otra posible implementación es la de un histórico de páginas visitadas.

- Patrón de selección de navegación, que proporciona al usuario un acceso rápido a las páginas/ubicaciones que presentan un mayor interés para él o que accede más a menudo. La implementación por defecto de este patrón consiste en la inclusión de una página abstracta y un mecanismo de adición/borrado de enlaces asociados a dicha página abstracta (*bookmark*).

- Patrones de Control de Comando. Por último, los patrones de control de comandos definen tanto las características de los caminos de acceso a la funcionalidad de la aplicación como el modo en que el usuario puede interactuar con dicha funcionalidad. Entre ellos, destacan:

  – Patrón de observación de comando: equivalente al patrón de observación de navegación. La implementación por defecto en OO-H consiste en la implementación de un histórico de comandos. Una implementación alternativa es la inclusión de mecanismos de tipo *deshacer* en la interfaz.

  – Patrón de confirmación de comando: causa que la interfaz solicite una confirmación explícita antes de invocar cualquier servicio que cause algún cambio en el estado del sistema. La implementación por defecto de OO-H consiste en la generación de una ventana de confirmación *javascript* ante la invocación de cualquier método de tipo *isQuery=false*.

  – Patrón de reintento de comando: permite al usuario reintentar la ejecución de un comando un número (limitado o ilimitado) de veces en caso que haya fracasado dicha ejecución. La implementación por defecto de OO-H consiste en la recarga de la página de invocación con la adición de un mensaje en caso de que el método haya devuelto algún tipo de información que pueda aclarar el motivo de fallo de la ejecución.

  – Patrón de anulación de comando: permite al usuario interrumpir la ejecución de un comando sin esperar a que el servicio

haya devuelto ningún valor a la interfaz, y continuar navegando por el sistema. La implementación por defecto de OO-H consiste en la adición de un botón *Anular* a todos los formularios de invocación de servicio.

El modelo de implementación web obtenido como resultado del proceso de generación del DPA a partir del DAN es utilizado en OO-H no sólo como base sobre la que aplicar distintos tipos de patrones, sino, sobre todo, como base sobre la que realizar la actividad de diseño de presentación, que es el último paso antes de poder proceder a la generación de cada versión de interfaz. A continuación presentamos el modo en que este diseño de presentación se realiza en OO-H.

8.4 El Diagrama de Diseño Visual

OO-H, al contrario que otras aproximaciones para el diseño de aplicaciones hipermediales, integra una actividad de diseño de interfaz que no se limita al esbozo de apariencia de las páginas, sino que permite describir sus características definitivas, que sólo tendrán que ser posteriormente traducidas al entorno de implementación deseado. Para la especificación de la apariencia, OO-H toma como entrada cada una de las páginas abstractas contenidas en el DPA, y organiza su contenido mediante un nuevo diagrama: el *Diagrama de Diseño Visual*(DDV) [17].

Esta organización se realiza en base a las características de presentación (ubicación de cada elemento, estilos, colores, etc.) asociadas por defecto a cada uno de los constructores del DAN durante el proceso de generación del DPA por defecto.

Volviendo a nuestro ejemplo, en la Fig. 8.6 podemos observar el DDV por defecto asociado a la página abstracta *Ocupación* (ver Fig. 8.2), cuya especificación textual fue presentada en la Fig. 8.3.

[17] El entorno de generación de OO-H garantiza que la apariencia final de dicha interfaz va a ser totalmente fiel a la especificación del DDV.

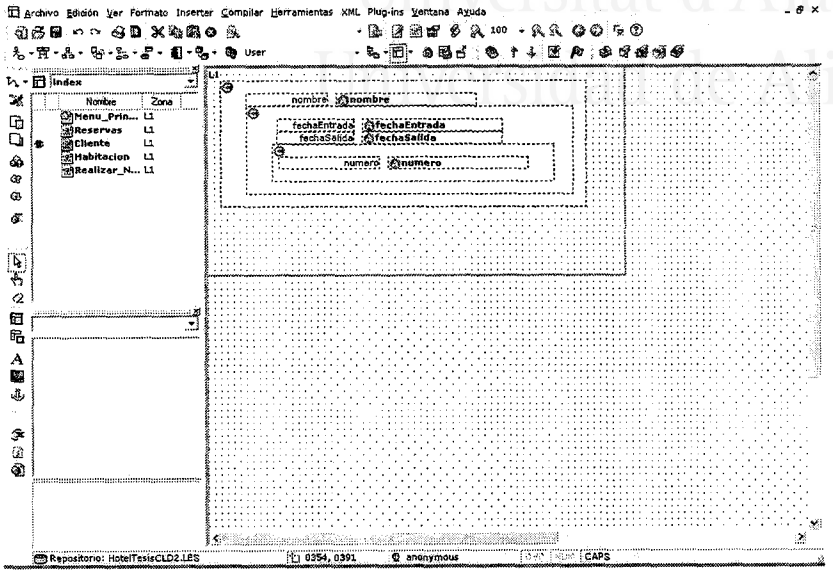

Figura 8.6. DDV por defecto asociado a la página abstracta *Ocupación*

En esta figura podemos observar cómo las distintas colecciones anidadas se presentan indentadas en la página (zonas etiquetadas con la letra *C*) mientras que los atributos correspondientes a cada objeto, todos ellos numéricos o cadenas, se han representado mediante cajas de texto.

A partir de estos DDV por defecto, OO-H permite cualquier tipo de modificación sobre la apariencia y contenido estático de la interfaz: composición de páginas abstractas en una sola página física, inclusión de imágenes, adición de etiquetas, etc. Todos estos refinamientos se realizan en el seno de un entorno de prototipado que presentaremos en el capítulo 9. Por ahora basta saber que este entorno incluye un conjunto de herramientas que permiten tanto el diseño de marcos para la visualización simultánea de varias páginas abstractas en una sola página física como la edición de las propiedades gráficas de cada uno de los elementos de las distintas páginas abstractas. En la Fig. 8.7 podemos observar un

posible resultado de aplicar dichos refinamientos sobre el DDV por defecto presentado en la Fig. 8.6.

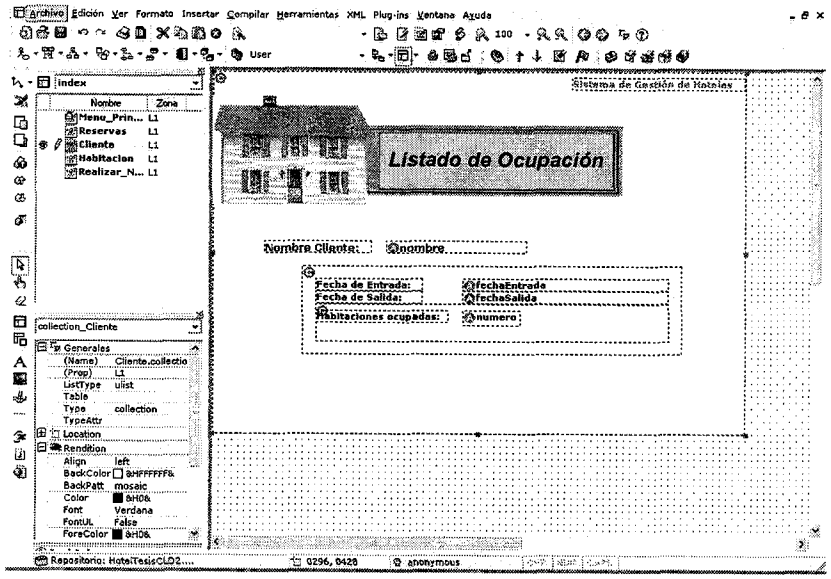

Figura 8.7. DDV refinado asociado a la página abstracta *Ocupación*

Para finalizar este capítulo nos gustaría destacar cómo el DDV, al igual que el DPA, se encuentra sincronizado con la especificación textual de la interfaz. De este modo, todos los cambios realizados a nivel visual se ven reflejados inmediatamente en las instancias de plantilla correspondientes. Además, y debido a la separación existente entre contenido, navegación y presentación, una regeneración del DPA a partir de un DAN modificado sólo causa la reescritura de las plantillas *tForm* y *tLink* (y no del resto de plantillas de presentación), por lo que los elementos que hayan permanecido invariantes seguirán asociados a los estilos, ubicaciones, etc. refinadas. Este hecho posibilita que el trabajo del diseñador gráfico se vea lo menos afectado posible ante una modificación de alto nivel de la interfaz, y constituye una de las principales ventajas de uso del DDV.

8.5 Conclusiones del Capítulo

Este capítulo ha presentado una taxonomía de plantillas que permite describir de una manera textual las interfaces modeladas en OO-H. Esta taxonomía, formada por seis tipos de plantilla (*tForm*, *tLink*, *tStyle*, *tLocation*,*tReference* y *tLayout*), incluye toda la información de los modelos de análisis y diseño de contenido y navegación, y los enriquece con consideraciones relativas a la presentación. Dicha taxonomía mantiene de este modo el grado de independencia entre los aspectos relativos al contenido, navegación y presentación de la interfaz, una de cuyas consecuencias más significativas es la posibilidad de conservación del trabajo realizado a nivel de presentación ante cambios producidos en niveles de abstracción superiores.

Además, esta descripción textual actúa de puente entre el espacio del problema y el espacio de la solución, al acercar la descripción del sistema a un modelo de implementación web basado en ficheros. De ellos, el conjunto de documentos instanciados a partir de la plantilla *tForm* y el documento instanciado a partir de la plantilla *tLink* permiten construir un Diagrama de Presentación Abstracta (DPA) que estructura de una manera visual las páginas que componen la interfaz y sus interrelaciones.

Aunque este modelo de implementación contiene toda la información necesaria para dar respuesta a los requisitos funcionales de usuario, a menudo es necesario refinarlo con características no capturables en modelos de más alto nivel. Para simplificar y agilizar dichos cambios OO-H proporciona un lenguaje de patrones estructurado en un Catálogo de Patrones de Interfaz, que recoge soluciones efectivas para problemas identificados hasta el momento dentro del ambiente web. Además, y con el fin de posibilitar su aplicación automática, cada implementación de patrón tiene asociada una regla de transformación que modifica la estructura del diagrama y/o el contenido de las plantillas. De este modo se facilita el reuso de experiencias de diseño y la consistencia tanto dentro de la aplicación como entre aplicaciones dentro de la organización.

Como última actividad de la fase de diseño, las páginas contenidas en el DPA pueden ser manipuladas de manera independiente con el fin de refinar su presentación. Esta manipulación se realiza mediante un nuevo tipo de diagrama, el Diagrama de Diseño Visual (DDV), que permite la edición de las propiedades gráficas de cada uno de los elementos de las distintas páginas abstractas.

En resumen, las contribuciones más relevantes de este capítulo son las siguientes:

- Presentación de una taxonomía de plantillas, definidas en XML, que separan los diferentes aspectos relacionados con la generación de las páginas de interfaz.
- Presentación de la semántica y generación por defecto del DPA a partir del DAN.
- Uso de un Catálogo de Patrones de Interfaz para la captura de características relevantes para la calidad del modelo de interacción usuario-sistema.
- Presentación del DDV para el refinamiento de la presentación de la interfaz.

9. El soporte CAWE de OO-H

Una aprendiz de carpintero puede querer sólo un martillo y una sierra, pero un artesano experimentado utiliza muchas herramientas de precisión. Del mismo modo la programación requiere sofisticadas herramientas para enfrentarse a la complejidad de las aplicaciones reales, y sólo la práctica con dichas herramientas proporcionará habilidad en su uso.

ROBERT L. KRUSE. DATA STRUCTURES AND PROGRAM DESIGN

Tal y como comentamos en el capítulo 2, la mayor parte de las propuestas hipermediales existentes en la actualidad reconocen la necesidad de un entorno de desarrollo que les dé soporte para incrementar sus probabilidades de éxito de implantación. OO-H no es ajeno a esta tendencia, y proporciona un entorno de desarrollo conocido como *Visual WADE*[1] (Gómez *et al.*, 2002; Cachero *et al.*, 2001a) que cubre todas las fases del método, e integra un compilador de modelos que permite la generación completa de las aplicaciones modeladas.

En este capítulo comenzaremos por justificar (ver sección 9.1) la necesidad de herramientas y compiladores de modelos como los proporcionados por OO-H para incrementar el nivel de implantación de procesos sistemáticos de desarrollo de aplicaciones hipermediales. A continuación (ver sección 9.2) describiremos las características generales del Visual WADE, entre las que destacan su extensibilidad y su soporte a la automatización de actividades. En la sección 9.3 nos centraremos en los mecanismos que esta herramienta proporciona para sustentar el proceso de modelado de OO-H que, como vimos en el capítulo 3, abarcan desde el análisis de requisitos hasta el diseño de presentación de la aplicación. En

[1] Web Application Development Environment

este contexto, un conjunto de actividades de evaluación (ver sección 9.3.3), que se ejecuta en el marco de un potente entorno de prototipado, permite la validación de la aplicación tanto por parte del diseñador como por parte del usuario final desde etapas muy tempranas del desarrollo. Una vez validada la aplicación, la etapa de construcción y adaptación de la aplicación en OO-H se apoya en un compilador de modelos. Este compilador permite, tal y como veremos en la sección 9.4, la generación de interfaces operativas, interconectables con módulos de documentación, módulos de lógica y bases de datos propietarios y/o generados por el propio compilador a partir de las especificaciones OO-H.

Visual WADE ha sido testada en entornos de producción reales. Destaca la experiencia con la empresa SUMA[2], que relataremos brevemente en la sección 9.5. Por último, las conclusiones del capítulo serán presentadas en la sección 9.6.

9.1 El impacto de las herramientas CAWE en la implantación de métodos hipermediales

En general, el desarrollo de aplicaciones en el seno de las organizaciones presenta un conjunto de ventajas respecto al uso de aplicaciones *off-the-shelf*, la externalización[3] o el uso de sistemas de planificación de recursos en la organización[4], que se presentan como alternativas al desarrollo de estos sistemas (Butler, 2000). Estos beneficios incluyen permitir la implantación de aplicaciones particulares que supongan algún tipo de ventaja competitiva, permitir la evolución y reingeniería de sistemas para cubrir necesidades cambiantes de negocio o dar la posibilidad de desarrollar aplicaciones que den respuestas mucho más específicas a las necesidades de la organización. En este sentido, la complejidad de las

[2] SUMA es el organismo autónomo de gestión tributaria de la Diputación de Alicante. Los lectores interesados pueden obtener más información acerca de esta organización en http://www.suma.es

[3] *outsourcing*

[4] ERP: Enterprise Resource Planning systems

aplicaciones demandadas y la creciente escasez de programadores en la actualidad requiere tanto la sistematización del proceso de desarrollo como la ayuda de entornos CASE[5] que agilicen y simplifiquen el diseño e implantación de la aplicación.

Sin embargo, como ya señalamos en el capítulo 2, estudios recientes (Barry & Lang, 2001) demuestran que, si nos restringimos al ámbito hipermedial, la implantación de técnicas y herramientas ingenieriles en los procesos de desarrollo web en el seno de las organizaciones es muy bajo. Este hecho contrasta con la actitud favorable que tienen los desarrolladores de este tipo de aplicaciones ante la adopción de algún tipo de metodología y/o herramienta como modo de mejorar aspectos como la velocidad o la calidad del proceso y del producto del desarrollo de sistemas (McClure, 1998; Butler, 2000).

Desde este trabajo se defiende la idea de que, con el fin de cambiar esta tendencia y facilitar la evaluación e implantación de procesos y técnicas de desarrollo sistemático como las presentadas en el capítulo 2, es imprescindible disponer de entornos integrados que les den soporte. En este sentido, el uso de entornos que apoyan el uso de notaciones y procesos de desarrollo de aplicaciones genéricas (e.g. UML, Proceso Unificado) como pueden ser Rational Rose (Rational Rose, 2002), Visible Analyst (Visible Analyst, 2002), Together (Together Soft, 2002) o Genova (Genova Project, 2002)[6], se han demostrado insuficientes para cubrir las necesidades del diseñador en el desarrollo de aplicaciones web.

Por tanto se hace necesario abordar el desarrollo de nuevos entornos CAWE, y para ello es necesario tener en cuenta un conjunto de características que se han demostrado básicas para el éxito de implantación de herramientas CASE genéricas, y que incluyen la simplicidad de uso, la inclusión de posibilidades de personalización, el soporte a múltiples plataformas y tecnologías y la capacidad de generación automática de código. Todas estas ca-

[5] Computer Aided Systems Engineering

[6] Genova es un plug-in de Rational Rose que permite la generación de interfaces de usuario a partir de modelos UML, así como parametrizar distintos aspectos de diseño.

racterísticas son tenidas en cuenta en Visual WADE, tal y como veremos a continuación.

9.2 Características generales de Visual WADE

Visual WADE es un entorno de desarrollo integrado de aplicaciones hipermediales que proporciona un soporte automatizado a las distintas actividades incluidas en el proceso de desarrollo de OO-H. Para ello, incluye un conjunto de elementos, que son:

- Capa conceptual: entorno de modelado del contenido, navegación y presentación de la aplicación.
- Capa externa: entorno de prototipado de aplicaciones.
- Capa de implementación: compilador de modelos que permite la generación de aplicaciones totalmente operativas y fieles a los modelos.
- Capa de ejecución: integración del producto con bases de datos, servicios preexistentes, etc.

La capa conceptual facilita la etapa de autoría mediante capacidades como son el cambio automático de vistas, sincronización y comprobación de modelos o automatización de partes del proceso de desarrollo. Tanto la actividad de modelado lógico de la presentación como las de evaluación del diseño se realizan en Visual WADE en el seno de un entorno de prototipado (capa externa), que permite no sólo modificar las páginas individuales de una manera visual, sino simular la ejecución de la aplicación mediante una opción de menú que emula los saltos de navegación a través de dichas páginas. Un compilador de modelos integrado en el entorno es capaz de generar interfaces de aplicación fieles a las especificaciones de diseño, así como bases de datos de apoyo, módulos de lógica de interfaz y módulos de documentación del sistema. Además, este compilador también es capaz de generar módulos mediadores que permiten su interconexión con bases de datos y módulos de lógica propietarios.

Con esta herramienta, el proceso de desarrollo de aplicaciones en OO-H puede ser definido como un desarrollo RAD[7], en el que un pequeño equipo es capaz de producir aplicaciones operativas en un período de tiempo medible en semanas o incluso días. Además, el uso extensivo de los prototipos que hace Visual WADE favorece la participación del cliente en el proyecto y le permite ver los cambios inducidos por sus sugerencias de manera prácticamente inmediata.

9.2.1 Extensibilidad de la herramienta

Visual WADE soporta todos los modelos de OO-H. Para ello, incluye un metamodelo implementado en Python[8] que almacena los distintos constructores, sus conexiones y sus propiedades.

Este metamodelo puede ser manipulado de una manera visual (mediante la adición de constructores o la edición de sus propiedades en el contexto de cada diagrama) o de una manera textual, mediante el intérprete Python incluido como parte del propio entorno de desarrollo. Cuando se arranca el entorno visual, éste carga a su vez la lógica Python que permite manipular el metamodelo. A partir de ese momento, todo el proceso de construcción de modelos es controlado mediante esta lógica. Por ejemplo, si se incurre en algún tipo de error sintáctico como puede ser un intento de conexión no permitida entre dos constructores, la lógica Python es la encargada de detectar el error e informar al diseñador.

[7] Rapid Application Development.

[8] Python es un lenguaje de programación orientado a objetos, portable e interpretado. Algunas de sus ventajas frente a otros lenguajes orientados a objetos bien conocidos (e.g. Java) son:

- Simplicidad (debida en parte a su alto nivel de abstracción)
- Facilidad de extensión mediante la adición de módulos implementados en un lenguaje compilado, como puede ser C o C++.
- Un compilador-intérprete que se embebe con facilidad en otras tecnologías, como puede ser Visual Basic, lenguaje en el que está implementado Visual WADE.
- Precio (es gratuito).
- Disponibilidad de sus fuentes en Internet.

Los lectores interesados pueden acudir a *http://www.python.org* para obtener una información exhaustiva acerca de este lenguaje.

Este tipo de arquitectura, junto al uso de un lenguaje interpretado, permite que la modificación de la lógica de manipulación del metamodelo (e.g. para refinar alguna regla sintáctica del método) pueda realizarse sin tener que recompilar ni entorno cliente ni el propio módulo de lógica.

Además, Visual WADE ofrece al diseñador una consola Python embebida en el entorno de modelado (ver Fig. 9.2). Este hecho no sólo aporta la posibilidad de manipular modelos dentro del entorno de una manera programática (y no sólo visual), sino que permite codificar, probar y almacenar procedimientos de extensión, mantenimiento, validación e incluso mejora automática de parámetros de calidad aplicables sobre los diagramas. Por ejemplo, Visual WADE dispone de un procedimiento que, cuando se aplica sobre el modelo de dominio, se asegura de que la forma de nombrar clases, atributos y métodos en el diagrama de clases cumpla la recomendación UML de escritura de nombres de clase que comiencen con mayúscula y nombres de atributos y métodos en minúscula. También dispone de un *plug-in* que permite la carga de diagramas de clase creados en *Rational Rose*. Nuevos *plug-ins* (ficheros con lógica Python) pueden ser añadidos sin más que ejecutar una opción del menú general de Visual WADE. Desde ese instante su contenido se ofrece al diseñador, que ni siquiera tiene que reiniciar la aplicación. Del mismo modo, modificaciones comunes de estructura/contenido de las aplicaciones (e.g. inclusión de una cabecera/pie de página en todas las páginas finales generadas) pueden ser programadas y almacenadas para ser reutilizadas en ocasiones posteriores y agilizar así el proceso de desarrollo[9].

Experiencias empíricas demuestran que el alto grado de flexibilidad que este mecanismo proporciona a Visual WADE para su adaptación a nuevas necesidades es una de las características más apreciadas del entorno en el mundo empresarial, tal y como veremos en la sección 9.5. Esta ventaja no se ve eclipsada, en contra

[9] Como ya vimos en el capítulo 8, este tipo de modificaciones se codifican en una *Regla de Transformación* que forma parte de alguno de los patrones contenidos en el Catálogo de Patrones de OO-H, y que determina los cambios que la aplicación de dicho patrón en el seno de Visual WADE introducen en la especificación de la interfaz.

de lo que podría pensarse, por ningún tipo de efecto observable que el uso de un lenguaje interpretado podría causar sobre el rendimiento de la aplicación. Actualmente la herramienta ha sido testada con un sistema que incluía diez mil instancias de clases, cada una con quince atributos, un constructor, un destructor y quince métodos arbitrarios, donde cada método incluía cinco argumentos. A pesar de su complejidad, ni el entorno gráfico ni las operaciones de interacción con el metamodelo acusaron ningún tipo de demora significativa[10].

9.2.2 Automatización de actividades

Como hemos podido observar a lo largo del presente trabajo, los efectos de las distintas actividades de modelado en OO-H se materializan en la creación y/o refinamiento de un conjunto de modelos que capturan aspectos relativos al contenido, navegación y presentación de la interfaz de la aplicación. Esta construcción y refinamiento de modelos en general puede abordarse con distintos grados de automatismo (Molina *et al.*, 2002):

- Refinamiento manual o poco asistido: decisiones creativas, basadas en la experiencia y pericia del diseñador. Resultados inciertos.
- Refinamiento semiautomático o asistido: asistentes, reglas de diseño, etc. guían al diseñador a la hora de tomar decisiones, restringiendo así el número de soluciones posibles. No obstante, sigue siendo el diseñador el que toma las decisiones pertinentes.
- Refinamiento automático: uso de técnicas de transformación que incluyen la toma de decisiones en función de una serie de parámetros del entorno. De este modo se posibilita la generación temprana de resultados tangibles, a costa de una menor flexibilidad en las soluciones obtenidas.

[10] El sistema fue testado en un Pentium III con 128 Mb de RAM bajo Windows 2000

Un resumen de las principales ventajas e inconvenientes de cada aproximación puede ser visto en la Fig. 9.1. En esta figura se observa cómo tanto el coste como el número de errores y el tiempo de prototipado en la fase de diseño disminuye en función del grado de automatismo con el que se refinen los modelos. Sin embargo, también disminuyen las posibilidades en cuanto al número y tipo de dichos refinamientos. Es por ello que Visual Wade ha adoptado una solución mixta: la información recogida en cada fase sirve de entrada para la generación de los diagramas por defecto de la fase siguiente (refinamiento automático). A partir de ahí el diseñador, ayudado por una serie de guías y patrones (de una manera asistida), puede continuar refinando los modelos.

	Manual	Semiautomático	Automático
Coste de Diseño	Alto	Medio	Bajo
Variantes de Diseño	Muchas	Muchas	Pocas
Errores de Diseño	Sí	No	No
Tiempo de Prototipado	Lento	Medio	Rápido

Figura 9.1. Características de las distintas aproximaciones para el refinamiento de modelos

A continuación veremos en más detalle el modo en que Visual WADE soporta el desarrollo de los distintos modelos que forman parte de OO-H.

9.3 Modelado de Aplicaciones en Visual WADE

Visual WADE permite tanto estrategias de desarrollo colaborativo, donde varios diseñadores trabajan al mismo tiempo sobre los mismos modelos, como estrategias de trabajo individual. Para ello, la herramienta dispone de un módulo cliente y un módulo servidor. El módulo servidor es el encargado de gestionar entornos de trabajo colaborativos y mantener un repositorio de proyectos

comunes, mientras que el módulo cliente contiene todos los elementos necesarios para el desarrollo de proyectos individuales. A partir de ahora nos centraremos en el módulo cliente, que es el que proporciona soporte a todas las características que incluye OO-H y que han sido presentadas a lo largo del presente trabajo.

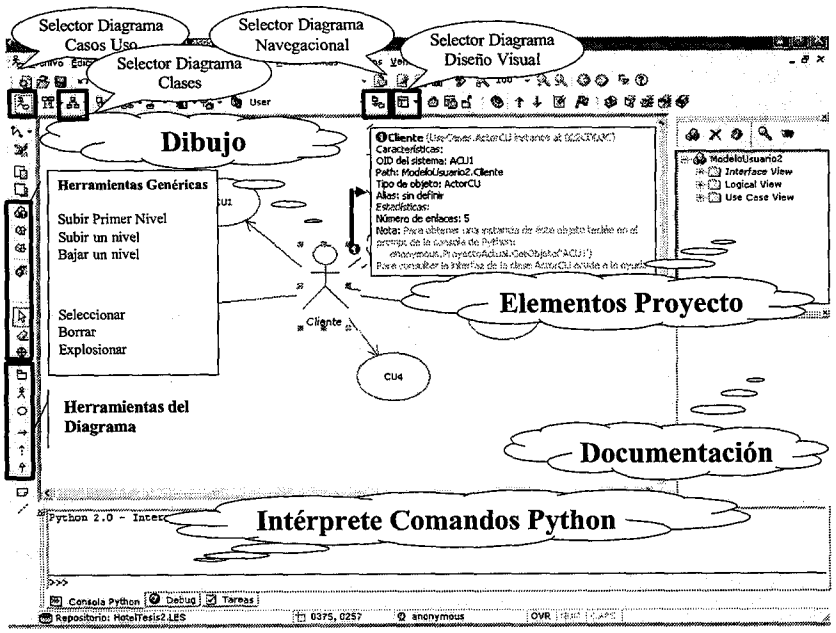

Figura 9.2. Entorno de trabajo de Visual WADE

El entorno de trabajo de Visual WADE presenta un aspecto como el que puede verse en la Fig. 9.2. En ella se puede observar una división en cuatro áreas: área de dibujo (donde se van diseñando los distintos diagramas), área de acceso a otros elementos del proyecto, área de documentación (donde el diseñador introduce los comentarios que considere oportunos acerca del diagrama en su conjunto o de cualquier elemento en particular) y consola de Python.

Cualquier elemento gráfico introducido en el área de dibujo dispone de un icono en su esquina superior derecha que proporciona

información bajo demanda sobre las propiedades del constructor en cuestión. Como ejemplo, en la Fig. 9.2 se puede observar la información proporcionada cuando se activa el icono de información correspondiente al *Cliente* del diagrama de casos de uso que aparece diseñado. Entre estas propiedades destaca el OID[11] con el que se ha almacenado el objeto en el metamodelo de OO-H, que permite al diseñador acceder y modificar el elemento de manera programática mediante el intérprete de comandos Python, tal y como comentábamos en la sección 9.2.

El cambio de contexto entre modelos se produce mediante la selección del icono correspondiente al diagrama que se desea diseñar en la barra de herramientas que aparece en la parte superior de la Fig. 9.2[12].

Por otro lado las principales herramientas genéricas de Visual WADE aparecen siempre situadas en la parte lateral izquierda del área de dibujo (ver Fig. 9.2). Los primeros tres iconos (tres cubos con una flecha hacia arriba, un cubo con una flecha hacia arriba y un cubo con una flecha hacia abajo) corresponden a herramientas para subir al primer nivel del diagrama, subir al nivel anterior o bajar al siguiente nivel, respectivamente. La *flecha* es el icono de selección por excelencia, y sirve para seleccionar o mover los distintos constructores del diagrama. El *borrador* se utiliza para borrar elementos del diagrama. Por último, la *diana* es utilizada en todos los modelos de OO-H para explosionar los paquetes y acceder a subniveles del diagrama.

Por debajo de estos iconos, otra barra de herramientas muestra los constructores correspondientes al tipo de modelo que estemos diseñando. Todas estas herramientas también se pueden seleccionar pulsando con el botón izquierdo del ratón en una franja vacía del área de dibujo y eligiendo la herramienta deseada del menú contextual que Visual WADE proporciona como resultado de esta

[11] Object IDentifier

[12] En esta barra de herramientas podemos ver cómo Visual WADE está concebido para dar soporte a la mayoría de diagramas estándares UML. Sin embargo, en el estadio actual de desarrollo, sólo los diagramas incluidos como parte de la propuesta OO-H están soportados por completo, y por tanto serán los que centren nuestra discusión a lo largo de este capítulo.

acción. La misma acción (botón izquierdo del ratón) sobre cualquier elemento del diagrama causa la edición en modo rápido del nombre del constructor correspondiente. Por último, si lo que se desea es acceder a las propiedades de dicho constructor se debe realizar con el ratón una doble pulsación sobre el elemento.

Este entorno proporciona soporte a las distintas actividades que conforman el proceso de desarrollo hipermedial de OO-H. Estas actividades incluyen, tal y como vimos en el capítulo 3, una actividad de análisis de requisitos, una actividad de ingeniería (análisis de dominio, análisis de navegación, diseño de dominio, diseño de navegación, diseño de presentación), una actividad de construcción y adaptación (integración, arquitectura de ejecución, configuración de parámetros de entorno, generación) y una actividad de evaluación (prototipado y validación).

A continuación veremos brevemente la integración en Visual WADE de cada una de ellas.

9.3.1 Soporte a las actividades de análisis de requisitos

La primera actividad a la que da soporte Visual WADE es a la introducción del Diagrama de Casos de Uso obtenido como fruto de la actividad de análisis de requisitos. En el estadio actual de desarrollo de la herramienta, este diagrama es utilizado como mera documentación del sistema y por tanto su soporte se limita a proporcionar los elementos necesarios para poder diseñarlo.

9.3.2 Soporte a las actividades de ingeniería

Por lo que respecta a las actividades de ingeniería, Visual WADE proporciona el soporte necesario para la construcción y evolución de cuatro diagramas: el diagrama de clases (DC), el diagrama de acceso navegacional (DAN), el diagrama de presentación abstracta (DPA) y el diagrama de diseño visual (DDV).

Estos diagramas tienen como principal característica la trazabilidad de muchos de los conceptos que incluyen. Por ejemplo, las

clases navegacionales del DAN son vistas sobre las clases de dominio, por lo que los nombres de los atributos/métodos del DAN que provengan del DC deben coincidir. Así, si se cambia un atributo del DC que aparezca en alguna clase navegacional, ésta se debe actualizar para mostrar ese nuevo nombre. Visual WADE asegura esta sincronización entre el modelo de dominio y el modelo navegacional (y, en general, entre todos los modelos) mediante la inclusión de un *Gestor de Eventos* que se encarga de ir apilando las acciones que deben ser propagadas mientras el usuario trabaja en un modelo, y de irlas desapilando según se va realizando dicha propagación. Así, si por ejemplo se produce un cambio en el nombre de un atributo, Visual WADE es la encargada de propagar ese cambio a los diagramas navegacionales que lo incluyen con el fin de evitar inconsistencias.

El resultado de las actividades de diseño de dominio y diseño de navegación puede ser compilado y dar lugar a una especificación XML que recoge las distintas páginas abstractas que conforman la interfaz, y que fueron comentadas en el capítulo 8. Una vez realizado este paso, cambios en el modelo navegacional (e.g. el cambio de un enlace en origen a un enlace en destino, lo que implica una redistribución de la información en páginas abstractas) requieren la recompilación explícita por parte del diseñador para que dichos cambios se vean reflejados en la especificación XML. Visual WADE asiste en esta recompilación, y emite un aviso cuando dicha recompilación es necesaria.

Es importante destacar que esta regeneración no implica una pérdida del trabajo que el diseñador haya podido realizar a nivel de presentación, sino que los cambios producidos por dicho trabajo (ya sean relativos a inserción, borrado de elementos o modificación de propiedades de estos elementos de presentación) se conservan y se aplican automáticamente sobre la parte que no ha variado de la nueva especificación de interfaz. Así, el diseñador sólo tendrá que refinar la presentación de los elementos que hayan aparecido como resultado de la regeneración.

El refinamiento de la navegación mediante patrones (realizado a nivel de DPA) y la actividad de diseño de presentación (reali-

zada en base a distintos DDV, uno para cada página abstracta del DPA) se maneja en Visual WADE mediante la manipulación directa de la especificación XML. En este caso la sincronización de modelos se manifiesta en que cualquier cambio efectuado en los diagramas queda inmediatamente reflejado en la especificación XML y viceversa, es decir, cualquier cambio realizado en la especificación XML queda inmediatamente reflejado en el diagrama. El editor de ficheros XML que incorpora Visual WADE puede ser visto en la Fig. 9.3.

Figura 9.3. Editor de ficheros XML de Visual WADE

En esta figura podemos ver cómo se ha editado la página de contenido que permite la edición de la especificación XML correspondiente a las reservas que cada cliente ha efectuado en el hotel. Esta página es una instanciación del DTD *tForm* que presentamos en el capítulo 8.

Aún más, cualquier cambio realizado sobre la especificación XML de la aplicación (a nivel de DPA o de DDV) se preserva cuando dicha especificación es regenerada (por haber variado aspectos a nivel de DC o de DAN). Esta preservación del trabajo de refinamiento y presentación de la aplicación favorece el desarrollo incremental de aplicaciones.

El compilador OCL. No podemos olvidar que uno de los mecanismos más potentes de los que dispone OO-H para la especificación de restricciones son los filtros que se asocian a los distintos elementos del método (atributos derivados, enlaces de navegación, valores por defecto de los parámetros, etc.). Para dar soporte a esos filtros y permitir tanto su verificación en tiempo de diseño como la generación de la consulta pertinente durante la etapa de construcción y adaptación, Visual WADE incorpora un compilador OCL propio que implementa la gramática presentada como parte de la especificación UML y valida las fórmulas en función de la información contenida en el modelo de referencia. Este compilador además ha sido extendido para dar soporte a todas las extensiones que presentamos en el capítulo 5, y que son:

- *?*: valor que debe ser introducido por el usuario de la aplicación en tiempo de ejecución.
- *valueIn...*: rango de valores que puede adquirir una variable introducida por el usuario.
- Operador *like* y carácter comodín *: permite la comparación de cadenas.

9.3.3 Soporte a las actividades de evaluación

OO-H, al contrario que otros métodos hipermediales, realiza las actividades de prototipado y validación en dos momentos del proceso de desarrollo: durante la actividad de ingeniería y tras la generación de cada versión de la aplicación. Durante la actividad de ingeniería es posible obtener y animar prototipos no funcionales en un entorno de prototipado que no requiere haber completado

los modelos, y que permite validar decisiones relativas al contenido, navegación y presentación de la aplicación. Por otro lado, una vez generada cada versión de la aplicación, se deben validar, además de estos rasgos, otros relativos a la parte dinámica de la aplicación: filtrado correcto de la información, interconexión con módulos externos, acceso a bases de datos, gestión correcta de permisos, manejo adecuado de la personalización, etc.

El prototipado de interfaces en la actividad de ingeniería. Partiendo de una posible estructura de navegación a través del espacio de información, obtenida como resultado de la actividad de diseño de navegación, Visual WADE es capaz de generar un prototipo no funcional de la interfaz modelada que puede ser navegado dentro del entorno[13].

La obtención de un prototipo desde fases tan tempranas del desarrollo es posible debido a que Visual WADE es capaz de automatizar, con la ayuda de un conjunto de plantillas predefinidas, todas las actividades relacionadas con el diseño de presentación: ubicación de elementos, estilos, colores, etc. La principal ventaja de este entorno de prototipado es que no exige que los modelos estén completos, lo que permite validar y realizar los cambios necesarios sobre las decisiones de diseño en etapas tempranas de desarrollo, con la correspondiente reducción de costes. Experiencias empíricas demuestran que posibilitando que tanto diseñadores como usuarios visualicen algún tipo de resultado antes de llegar a la fase de generación y/o implementación se reduce el número de iteraciones necesarias para modelar la interfaz final, y por tanto se aumenta la eficacia del método.

Una imagen del entorno de prototipado de Visual WADE, junto con un ejemplo de página prototipada directamente a partir del DAN presentado en la Fig. 5.11, y que, recordemos, permitía obtener información acerca de la ocupación de nuestro hotel, puede ser vista en la Fig. 9.4. En ella observamos un área de visualiza-

[13] Recordemos que, tal y como comentamos en la sección 3.4.4, hablamos de prototipo no funcional porque no incluye datos reales ni conexiones con bases de datos o componentes externos, sino que se limita a mostrar las decisiones adoptadas hasta el momento sobre el contenido, la navegación y la apariencia de la interfaz.

ción donde se colocan todos los elementos de contenido con sus correspondientes etiquetas y una ubicación, color, fuente, etc. por defecto. Un visor que lista las páginas generadas a partir del DAN (las mismas que se representan en el DPA por defecto) permite la visualización/edición de cada una de ellas. Un icono representando un *ojo* al lado del nombre de la página indica que ésta es la página que está siendo visualizada en este momento. Si además aparece un *lápiz*, las características de cualquiera de sus elementos pueden ser modificadas a través de la *ventana de propiedades* que proporciona el entorno. Además, el entorno permite la adición de nuevos elementos a las distintas páginas para aumentar su impacto visual: imágenes, animaciones, etc. Por último, una herramienta de navegación permite simular los saltos de navegación a través de los enlaces contenidos en las páginas y comprobar la corrección de la estructura navegacional.

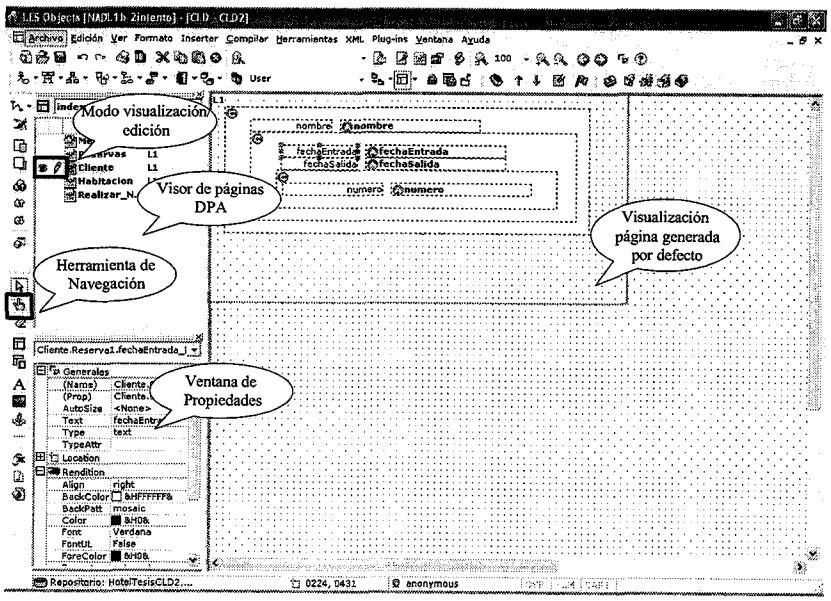

Figura 9.4. Entorno de Prototipado de Visual WADE

Validación. El prototipado permite una validación de la interfaz tanto por parte del diseñador como por parte del usuario modelada, y se centra en comprobar que todos los requisitos previstos están cubiertos y que el contenido, la navegación y la presentación son las adecuadas. Sin embargo, el entorno de prototipado no permite comprobar que efectivamente estos requisitos se van a ejecutar correctamente en la interfaz final. Es por ello que, una vez generada, la actividad de validación se debe volver a realizar de manera mucho más exhaustiva[14]. Esta nueva comprobación sobre el código generado debe incluir tanto aspectos propios de la aplicación (e.g. comprobar que se han definido correctamente los filtros de navegación, de manera que se selecciona correctamente la información destino de cada enlace) como cuestiones relativas a la compatibilidad del código con los distintos navegadores, rapidez, eficiencia...

Tal y como comentamos en la sección 9.2, para la generación de este código final Visual WADE incorpora un compilador de modelos que toma como entrada la especificación XML obtenida como resultado de la ejecución de las distintas actividades de ingeniería. Este compilador contiene el conocimiento necesario para generar una interfaz web operativa para la plataforma deseada. A continuación presentamos las principales características de este compilador de modelos.

9.4 Generación de aplicaciones en Visual WADE: el compilador de modelos

Una vez disponemos de toda la información necesaria, a la hora de abordar la construcción de la aplicación nuevamente se puede optar por tres tipos de aproximaciones (Molina *et al.*, 2002):

[14] En el estadio actual de desarrollo de Visual WADE, ésta no aporta ningún mecanismo de ayuda a esta actividad. La inclusión de una serie de criterios de calidad que puedan ser comprobados de manera automática y simplifiquen de este modo la validación de modelos queda por tanto como una propuesta de trabajo futuro.

- Programación Manual: Es la aproximación más utilizada, aunque también es la más lenta y la más sujeta a errores. En esta aproximación los modelos se convierten en meros documentadores del sistema, e incluso este papel se puede ver menoscabado por el mayor o menor grado de fidelidad de los programadores a la hora de seguir sus especificaciones. Requiere una fase de depuración y pruebas exhaustiva.
- Reutilización de Código: Esta aproximación está en la base de propuestas como el CBD[15]. Su principal ventaja es que permite reutilizar código que ha sido ampliamente testado y validado. Sus principales inconvenientes son la necesidad de tenerlo perfectamente catalogado (uso de repositorios) y la dificultad de encontrar rutinas de un nivel de abstracción elevado que se adapten de manera perfecta a los requisitos de la aplicación actual.
- Generación de Código: Por último, las técnicas de transformación y generación de código pueden ser utilizadas para, a partir de los modelos existentes, generar una implementación que se adapte perfectamente a las especificaciones. Además, esta aproximación evita gran parte de los errores de codificación. Su principal inconveniente es su falta de flexibilidad, debido a la adopción de decisiones relativas a la codificación de los aspectos modelados de manera automática[16]. Esta carencia se puede paliar parcialmente mediante el uso de traductores parametrizados, que permiten introducir variaciones en la configuración, arquitectura, lenguajes, etc.

Un tabla comparativa que recoge estas ventajas y desventajas de las distintas aproximaciones puede ser vista en la Fig. 9.5.

Visual WADE ha optado nuevamente por una aproximación mixta. Así, la herramienta proporciona un conjunto de traductores parametrizados que posibilitan una generación automática de la interfaz. Sin embargo, estos traductores no se limitan a generar

[15] Component Based Development.

[16] Obviamente, mientras más detallados sean los modelos menor será la necesidad de adopción de este tipo de decisiones automáticas.

	Programación Manual	Reutilización Código	Generación Código
Velocidad de Prototipado	Lenta	Media	Alta
Errores en el código	Muchos	Algunos	Pocos
Calidad de Ajuste	Depende programador	Depende programador	Total
Flexibilidad	Alta	Media	Baja

Figura 9.5. Características de las distintas aproximaciones para la generación de la aplicación final

las páginas (estáticas o dinámicas) que conforman dicha interfaz y los módulos que dan soporte a la navegación a través de ellas, sino que están preparados para la generación de módulos de conexión con servicios web externos, aspecto que introduce una estrategia de reutilización de código. En este sentido cabe destacar que, puesto que OO-H presupone que estos servicios se ofertan mediante interfaces estándares de servicio web, sus modelos se limitan a incluir la interfaz de los distintos módulos, y por tanto se eliminan los problemas de calidad de ajuste entre modelos e implementación final que la reutilización de código puede causar.

La recogida de parámetros se produce en OO-H a lo largo de tres actividades que forman parte de la actividad de construcción y adaptación del método. Éstas son:

- Configuración de la arquitectura de la aplicación. Para cubrir esta fase Visual WADE solicita parámetros relacionados con el tipo de arquitectura deseada, y que puede ser *cliente ligero*, *cliente pesado* o *aplicación distribuida* (Conallen, 1999). En la actualidad Visual WADE sólo soporta la generación de aplicaciones de tipo *cliente ligero*, aunque se prevé su extensión para otros tipos de arquitecturas.
- Integración. Durante el desarrollo de esta actividad se solicitan parámetros relacionados con la integración de la interfaz con módulos externos. Este punto será explicado con más detalle en la sección 9.4.5.
- Configuración de parámetros de entorno. Esta actividad implica la recogida de parámetros genéricos de generación de los módulos pertinentes: lenguaje cliente y servidor, uso o no de *cookies* en la interfaz generada, etc.

Estos datos se complementan con la descripción XML obtenida como resultado de la actividad de ingeniería y con un conjunto de plantillas que contienen conversiones estándar entre constructores abstractos de presentación y constructores físicos, dependientes del lenguaje de implementación final. Con esta información se puede proceder a la ejecución de la última fase de la actividad de construcción y adaptación: la generación propiamente dicha.

Es importante destacar cómo la arquitectura de la herramienta facilita la integración de nuevos compiladores sin necesidad de recompilar el entorno. Aunque la realización de un nuevo traductor (para Visual WADE o para cualquier otro entorno) puede resultar más compleja (en función del tipo de aplicación a migrar) que la migración manual de una aplicación a nuevas plataformas/dispositivos de acceso, el compilador queda disponible dentro del entorno, y por tanto a partir de ese momento cualquier otra aplicación que deba ser migrada puede serlo con un coste mínimo.

El resultado de esta estrategia es la generación de uno o varios de los siguientes componentes[17]:

- Una bases de datos.
- Una capa de aplicación que contiene: (1) lógica de conexión con módulos externos (servicios y bases de datos propietarias), (2) un módulo de lógica que recoge el conjunto de operaciones CRUD que pueden ser aplicadas sobre dicha base de datos (encapsuladas o no en forma de procedimientos almacenados) y (3) un módulo de lógica de interfaz (implementación de filtros, manipulación de las bases de datos generadas desde Visual WADE, etc.).
- Un conjunto de páginas interconectadas que constituyen la interfaz de la aplicación, y desde las que se tiene acceso a la capa de aplicación.

[17] La generación de cada uno de estos componentes será tratada en más profundidad en las secciones 9.4.1 a 9.4.4.

- Un conjunto de páginas que contienen la documentación del sistema.

Antes de pasar a detallar cómo se genera cada uno de estos componentes, nos gustaría destacar cómo la separación explícita del resultado de modelar la interfaz (capturada mediante la descripción XML de la misma) y los distintos compiladores de modelos (para distintas plataformas) en Visual WADE permite la reutilización de estos compiladores con otras propuestas hipermediales, siempre que éstas sean capaces de proporcionar una especificación XML de sus modelos compatible con el formato de entrada XML de los compiladores de OO-H, cuyos DTD's fueron presentados en el capítulo 8.

9.4.1 Generación de la Base de Datos

El primer elemento que debe ser generado en Visual WADE es la capa de persistencia de la aplicación. Esta capa de persistencia consiste en el estadio actual de desarrollo de la herramienta en una base de datos relacional que cumple dos objetivos:

- Mantener las estructuras que proporcionan soporte al propio entorno de ejecución de las aplicaciones generadas mediante Visual WADE: estructuras para la gestión de contexto de navegación, estructuras que posibilitan la personalización, etc.
- Si no existe una base de datos de dominio propietaria, proporcionar las estructuras necesarias para el almacenamiento y manipulación de clases, atributos y relaciones.

Para la consecución del primer objetivo, las estructuras que dan soporte al entorno de ejecución de la aplicación se generan a partir de un subconjunto de características del metamodelo de OO-H. Como ejemplo, todas las estructuras de soporte a las estrategias de personalización admitidas en OO-H se generan a partir de la vista de personalización del modelo de referencia que presentábamos en la Fig. 7.3. Un conjunto de transformaciones estándares, bien

documentadas (Pang, 1998), permiten la transformación de esta especificación OO a un modelo de base de datos relacional[18]. Del mismo modo, si es necesario generar las estructuras correspondientes al dominio de la aplicación (consecución del segundo objetivo), Visual WADE se basa en el diagrama de clases obtenido en la fase de diseño de dominio, y sobre él aplica el mismo tipo de reglas de transformación para obtener su esquema relacional.

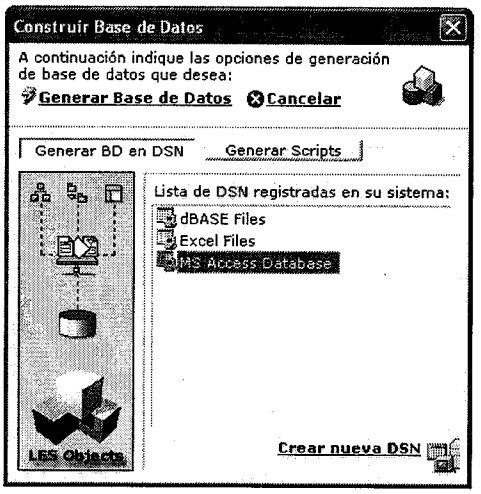

Figura 9.6. Parámetros de generación de la base de datos de la aplicación

En este momento Visual WADE es capaz de generar, tal y como vemos en la Fig. 9.6, una base de datos accesible mediante un nombre de fuente de datos (DSN[19]) o un guión que contiene sentencias SQL estándares de creación de tablas. Este guión permite la creación de la BD en cualquier SGBD no soportado directamente por el entorno.

[18] Una explicación más exhaustiva del modo en que esta transformación se realiza en Visual WADE puede ser consultada en (Párraga *et al.*, 2002).

[19] *Data Source Name*

9.4.2 Generación de la capa de aplicación

Por otro lado, la generación de la capa de aplicación también sirve a dos objetivos claramente diferenciados:

- Soporte a la lógica de navegación: materialización de las fórmulas OCL contenidas en los distintos diagramas del método. Los filtros OCL se traducen en sentencias de selección sobre los objetos del dominio, por lo que este módulo debe tener acceso a la base de datos de dominio.
- Soporte a la invocación de servicios: generación de módulos de conexión con módulos de lógica externos y/o de sentencias SQL (encapsuladas o no en procedimientos) de manipulación de los datos de dominio[20]. En la actualidad sólo se soporta la invocación de servicios externos mediante la generación de una llamada SOAP-RPC sobre HTTP, por lo que los servicios deben ser ofertados como Servicios Web para poder proceder a su invocación.

Tal y como podemos observar en la Fig. 9.7, para cubrir estos objetivos, Visual WADE genera un módulo *Mediador de Objetos* que recoge la petición de la interfaz. Esta petición puede consistir en una travesía de un enlace de navegación (posiblemente con algún tipo de filtro asociado) o en la invocación de un servicio de lógica, en cuyo caso es posible que la interfaz haya enviado, junto con la petición de servicio, un subconjunto de parámetros de dicho servicio. En cualquier caso, la primera tarea del mediador de objetos consiste en comprobar si el usuario tiene permiso para realizar la operación solicitada.

Además, si la navegación/servicio se realiza sobre algún objeto (en cuyo caso, y por motivos de seguridad, sólo se permite la invocación si el objeto reside en memoria), el mediador es el encargado de recuperar el/los objetos involucrados, y de pasárselos

[20] Recordemos que Visual WADE sólo genera aplicaciones completas si su sencillez permite que sus requisitos puedan ser cubiertos haciendo uso exclusivo de operaciones CRUD sobre una base de datos.

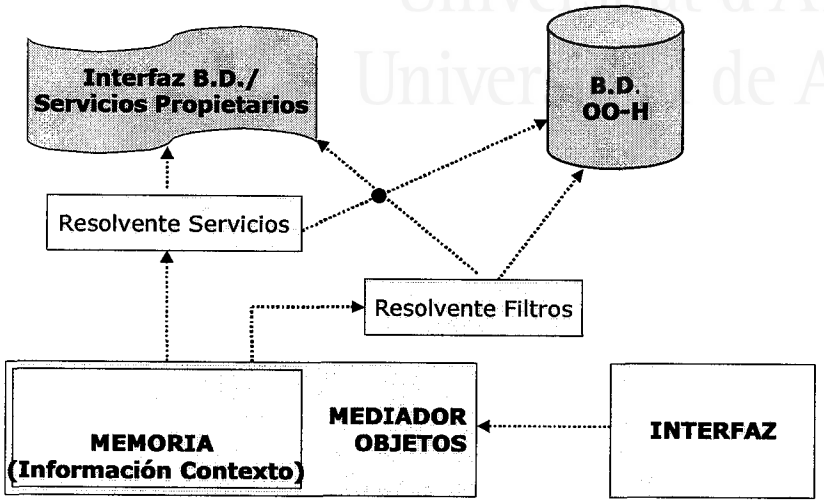

Figura 9.7. Arquitectura de la Capa de Aplicación Generada por el compilador de modelos de Visual WADE

al módulo resolvente correspondiente, junto con cualquier otro tipo de información de contexto (y los valores de los parámetros si se trata de un servicio).

Si la petición de la interfaz es de tipo navegacional, se invocará el *Resolvente de Filtros*, mientras que si es de tipo operacional se invocará el *Resolvente de Servicios*. Estos dos módulos son los encargados de interactuar con la base de datos generada por el propio compilador (que, recordemos, puede incluir o no estructuras de dominio) y/o con bases de datos y módulos de lógica externos. En cualquier caso, una vez realizada la operación deseada y recuperado el control por el *mediador de objetos*, éste manipula los resultados obtenidos (si existen) y rellena una estructura de tipo *ConjuntoDeResultados*, de tipo jerárquico. Esta estructura es devuelta a la interfaz, que se encarga de mostrarla tal y como le ha sido devuelta. De este modo se consigue una total independencia de la interfaz respecto a la arquitectura de la capa de aplicación.

Los parámetros necesarios para la generación del mediador de objetos y sus submódulos (resolvente de filtros y resolvente de servicios) son, tal y como vemos en las Figs. 9.8 y 9.9, el lenguaje de programación en que deseamos generar el módulo[21] y los parámetros de conexión con la base de datos generada por el propio compilador de Visual WADE.

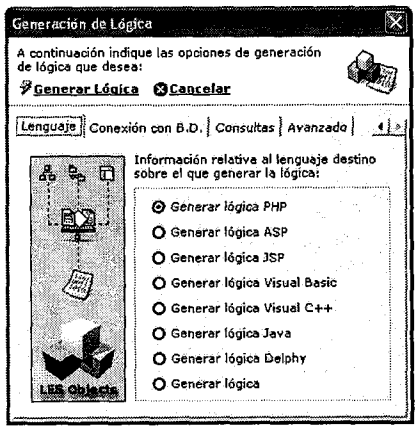

Figura 9.8. Parámetros de generación de la capa de aplicación: lenguaje de programación objetivo

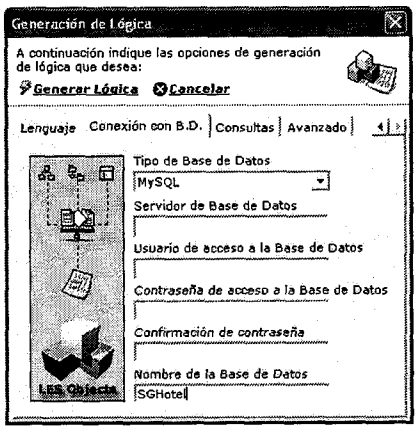

Figura 9.9. Parámetros de generación de la capa de aplicación: conexión con BD de Visual WADE

9.4.3 Generación de la interfaz

Una vez generada la base de datos y la capa de lógica OO-H, el compilador de modelos está en disposición de generar una interfaz operativa basada en estos elementos. Para ello, Visual WADE solicita un conjunto de parámetros que definen características generales de dicha interfaz que, como podemos observar en las Figs. 9.10 y 9.11, incluyen la ubicación y el lenguaje objetivo de las páginas generadas.

[21] Aunque el entorno está preparado para generar en un número indeterminado de lenguajes, Visual WADE sólo soporta por el momento la generación de lógica en PHP (PHP, 2002).

Esta información es útil para seleccionar el tipo de transformación que se debe realizar entre los constructores de interfaz abstractos representados en el modelo de diseño de presentación[22] y los constructores físicos particulares al lenguaje de implementación deseado. La equivalencia entre constructores para cada lenguaje de implementación soportado por Visual WADE se almacena en una plantilla XML que forma parte del entorno de compilación.

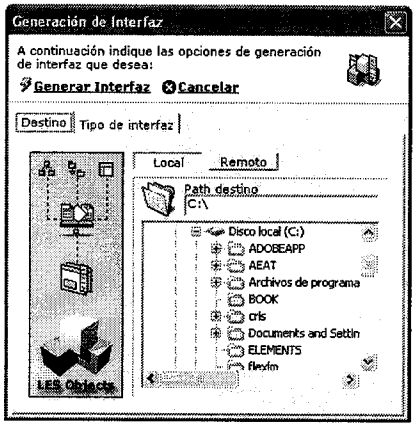

Figura 9.10. Parámetros de generación de la interfaz: ubicación física de las páginas

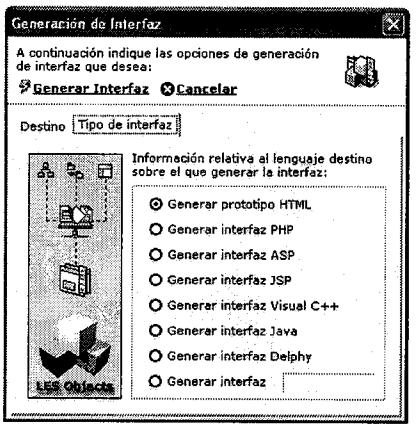

Figura 9.11. Parámetros de generación de interfaz: lenguaje de implementación

9.4.4 Generación de la documentación

Por último, como ya hemos visto a lo largo del capítulo, Visual WADE proporciona distintos mecanismos de documentación, a saber:

- Los propios modelos.
- Registro de los patrones que han sido aplicados en cada modelo.

[22] *widgets*

- Mecanismos de anotación que permiten la especificación, en lenguaje natural, de comentarios adicionales sobre los modelos y/o sobre cualquiera de sus constructores.

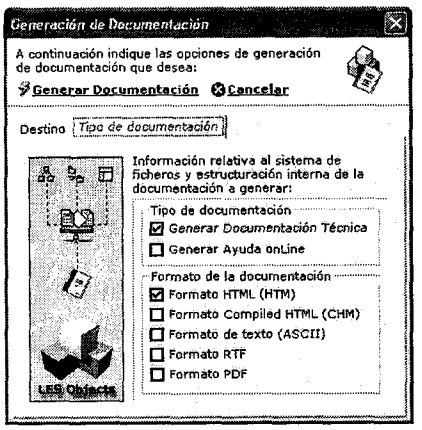

Figura 9.12. Parámetros de generación de la documentación: tipo y formato

Toda esta información puede ser accedida por el compilador de modelos, que de este modo es capaz de generar un módulo de documentación totalmente sincronizado con la implementación generada. En la Fig. 9.12 vemos cómo nuevamente Visual WADE permite la generación de distintos tipos de documentación (documento técnico o ayuda en línea) y cómo además esta información puede ser obtenida en distintos formatos: PDF, HTML, etc.

9.4.5 La integración de Servicios Web en las interfaces OO-H

Aunque aún se encuentra en un estadio de desarrollo muy básico, Visual WADE está siendo extendido para el soporte de invocación de servicios y/o bases de datos propietarias via SOAP-RPC [23])

[23] El espacio de nombres correspondiente a SOAP-RPC en su versión 1.2. es http://www.w3.org/2001/06/soap-envelope. Del mismo modo, el estilo de codificación corresponde a la especificación http://www.w3.org/2001/06/soap-encoding

sobre HTML[24]. Para la generación de los mensajes de llamada, Visual WADE utiliza la descripción de la interfaz de servicio capturada en el modelo de dominio, y respeta tanto los nombres de los parámetros como su orden.

Las razones que han motivado esta decisión están en la propia naturaleza del objetivo con el que nació OO-H, que recordemos es permitir el diseño no sólo de aplicaciones hipermediales intensivas en datos (donde la capa de lógica se reduce a operaciones CRUD sobre una base de datos) sino de interfaces hipermediales que puedan conectarse con bases de datos y aplicaciones propietarias. Esta consideración explícita de bases de datos y/o módulos de lógica propietarios claramente implica un entorno de ejecución distribuido para la aplicación resultante. Muchas tecnologías distribuidas exigen, para permitir la comunicación entre dos sistemas diferentes, que la misma tecnología de comunicación esté presente en ambos lados de la comunicación (e.g. DCOM (DCOM, 2002)), lo que claramente las inhabilita para su uso en OO-H, que no tiene control sobre la tecnología de los módulos que deben ser integrados. Otras, aunque sí permiten la interoperabilidad de aplicaciones ubicadas en distintas plataformas y desarrolladas con distintos lenguajes, no son capaces de trabajar con cortafuegos[25] (e.g. CORBA (OMG, 2002)).

De entre aquéllas que sí han resuelto estos problemas, XML y SOAP se han convertido en el estándar de facto para la interconexión de sistemas, y es por ello que han sido la tecnologías elegidas para materializar la integración de lógica de negocio en OO-H, mediante el uso de Servicios Web.

Esta decisión (frente a la integración de componentes tradicionales siguiendo una filosofía CBD) aporta varias ventajas, entre las que destacamos:

[24] En estos momentos el soporte se limita a la generación del mensaje SOAP y la recuperación de resultados. La integración de otras tecnologías como UDDI, que permite una integración *just-in-time*, queda por tanto pospuesta para versiones futuras de la herramienta

[25] firewalls

- El nivel de abstracción: los servicios web tienen un nivel de abstracción superior a los componentes: un servicio puede proporcionar acceso a un método sobre el que no se establece ningún tipo de prerrequisito de implementación interna: puede ser que esté implementado mediante una arquitectura basada en componentes, pero también puede, por ejemplo, proporcionar acceso a una BD. Así, los componentes proporcionan un tipo de elemento que puede ser utilizado en las implementaciones de servicios web.
- La disponibilidad: los servicios web presuponen que la lógica subyacente ya está disponible para su uso, y que sólo hay que conectarse a algún lugar, mientras que los componentes siguen una filosofía de tipo *instala y usa.*
- La ubicuidad: mientras que los componentes son mayormente utilizados dentro del ámbito de una empresa, los servicios web son especialmente útiles para federar sistemas pertenecientes a distintas empresas, y su acoplamiento es aún más bajo que en el caso de los componentes, debido a que se basan en estándares industriales y protocolos basados en Internet.

No obstante, somos conscientes de que los servicios web no son una panacea: quizás su mayor inconveniente es la inmadurez de esta tecnología: aspectos como seguridad, manejo de transacciones, etc. están sólo parcialmente resueltos, y se prevé la aparición y rápida evolución de nuevos estándares que aborden cada uno de los problemas detectados (Herzum, 2001). Nuevamente la extensibilidad de nuestra herramienta nos hace afrontar con confianza estos retos.

9.4.6 Arquitectura de la aplicación generada

Tal y como comentamos al comienzo de la presente sección, la arquitectura de las aplicaciones generadas mediante Visual WADE está actualmente predefinida como *thin client*, y presenta la estructura que fue explicada en la sección 3.6, y que reproducimos en la Fig. 9.13.

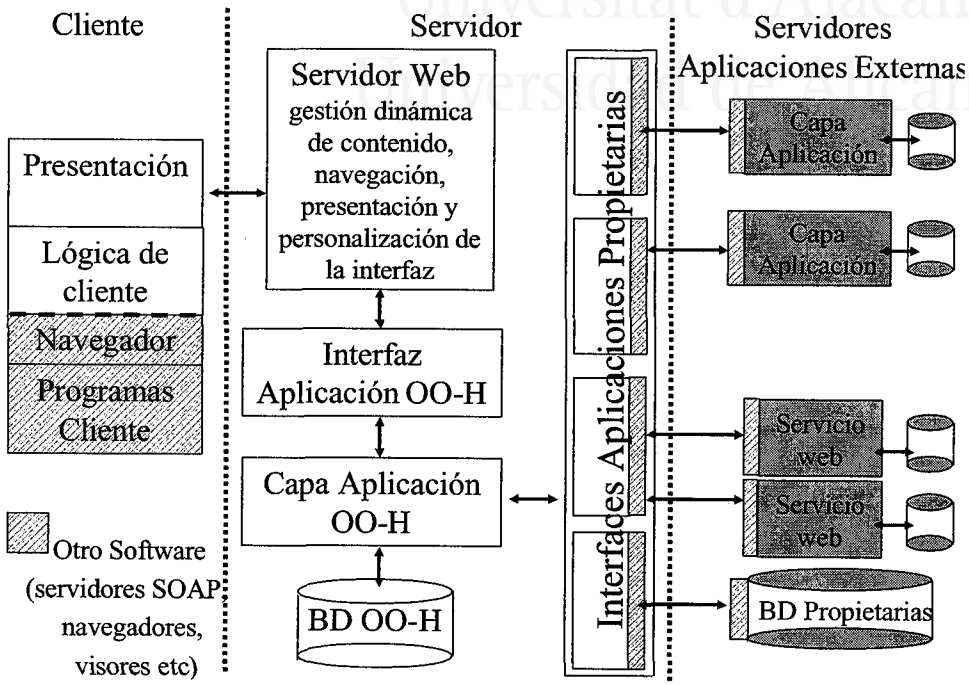

Figura 9.13. Arquitectura de las aplicaciones generadas mediante OO-H

Dentro de esta arquitectura, el compilador permite la generación de lógica de cliente *JavaScript* que da soporte a la validación de campos de formularios para los dos navegadores de uso más extendido: *Netscape* e *IExplorer*. Estos navegadores pueden además incorporar *plug-ins* o cualquier otro tipo de programa cliente que extiendan su funcionalidad. La comunicación entre los componentes cliente y servidor de la aplicación resultante se realiza sobre HTTP, y a la hora de generar las páginas que componen la interfaz de cliente, Visual WADE soporta dos opciones: generación de páginas estáticas (en XHTML[26]) o generación de páginas dinámicas (en PHP).

Estas páginas de interfaz están interconectadas con una capa de aplicación cuya descomposición interna fue explicada en la sección

[26] Xml HyperText Markup Language

9.4.2, y que proporciona acceso tanto a una base de datos de soporte generada por el propio compilador como a servicios y bases de datos propietarias.

La base de datos OO-H es de tipo relacional, y se deriva de los diagramas de clase que definen tanto el modelo de dominio como el metamodelo de OO-H.

Por último, en esta arquitectura, la inclusión de servicios web es el primer paso hacia el soporte a un nuevo paradigma, conocido como *Arquitectura Orientada al Servicio*[27]. El objetivo principal de SOA es permitir que los componentes residentes en Internet puedan ser publicados, invocados y descubiertos por los demás, lo cual aumenta la flexibilidad y evolución de los sistemas. En general, coincidimos con (Rodríguez & Díaz, 2002) en que ya se está produciendo una evolución de las aplicaciones web hacia arquitecturas orientadas al servicio, y no a los datos, lo cual exige una evolución en las aproximaciones de modelado para dar soporte a este cambio de perspectiva.

9.5 Implantación de Visual WADE: una experiencia empírica

Los que conformamos el equipo de desarrollo de Visual WADE somos conscientes de las dificultades que presenta la implantación de una CAWE en un entorno empresarial, debido a una multitud de factores contextuales que muchas veces dificultan la explotación de sus ventajas potenciales. Estos factores contextuales incluyen la actitud de los actores que intervienen en la implantación, factores organizacionales y factores de entorno (Butler, 2000). Entre los factores relacionados con los actores del proceso de implantación destaca la existencia de un estilo de gestión de proyectos participativo y la selección cuidadosa de los participantes en el equipo de proyecto. Los factores organizacionales incluyen la actitud positiva hacia la adopción de este tipo de herramientas por

[27] Service Oriented Arquitecture (SOA) (S. M. Vivekanandan, 2002)

parte de los directivos y la experiencia previa de la organización con herramientas similares. Por último, los factores de entorno incluyen aspectos como un buen plan de formación en el uso de la CAWE o la disponibilidad de un experto en la herramienta para ir resolviendo todos los problemas que puedan ir surgiendo durante el desarrollo del proyecto.

Sin embargo, consideramos imprescindible que cualquier herramienta de las características de Visual WADE sea testeada en un entorno empresarial. Es por ello que, teniendo en cuenta todos los factores mencionados, durante el último año se ha llevado a cabo una experiencia piloto en la que cinco personas de la empresa SUMA han abordado la construcción de aplicaciones propietarias con Visual WADE. Estas personas tenían un alto nivel de motivación, así como experiencia previa en la construcción de aplicaciones web. Los directivos de la empresa apoyaron expresamente el proyecto, y han dispuesto numerosos seminarios en los que estas cinco personas se han ido familiarizando con los distintos aspectos de la herramienta. Además, una persona del equipo de desarrollo de Visual WADE ha estado continuamente disponible para solucionar cualquier duda o problema que haya podido surgir.

De los comentarios e impresiones recogidos como resultado de esta experiencia, nos gustaría destacar cómo la potencia del entorno de prototipado de Visual WADE y su herramienta de navegación se ha demostrado clave: las personas involucradas en el proyecto han visto como una gran ventaja la posibilidad de propagar cambios en fases tempranas del proyecto hasta el producto final, y ver de manera inmediata los efectos de dichos cambios. Este aspecto les ha animado a refinar modelos y a obtener productos de una mayor calidad y con unos requisitos de mantenimiento mucho menores. Otra ventaja percibida por el personal de SUMA ha sido la integración de todas las herramientas necesarias para ir desde el análisis hasta la implementación final, que ha simplificado su percepción del proceso de desarrollo.

Esta experiencia ha supuesto el enriquecimiento de la herramienta con nuevos rasgos no soportados cuando se inició el proyecto. La

rapidez de implantación de dichos rasgos han demostrado el alto grado de flexibilidad de Visual WADE para su adaptación a nuevas necesidades empresariales. Esta característica ha permitido la introducción de todos los refinamientos sobre la aplicación con anterioridad a la fase de generación de código. De este modo se ha evitado la desincronización de modelos e implementación final, y el mantenimiento de las aplicaciones se ha convertido en una tarea trivial. Este hecho a su vez ha afianzado la confianza de SUMA en que Visual WADE sea capaz de dar respuesta a sus requisitos, y ha sido un factor clave para la renovación del convenio.

Por último, destacar que no ha existido una resistencia por parte ni de los analistas ni de los programadores en el uso de la herramienta. Por el contrario, los programadores, cuyo trabajo puede parecer más amenazado por el uso de una herramienta, celebraron la posibilidad de extender sus competencias y pasar a realizar actividades de análisis y diseño.

9.6 Conclusiones del capítulo

Este capítulo ha presentado las características más importantes de Visual WADE, el entorno CAWE qe da soporte al proceso de desarrollo de OO-H y permite el desarrollo rápido de aplicaciones web operativas[28].

Estas herramientas deben cubrir una serie de requisitos, entre los que la facilidad de uso, el proporcionar capacidades de personalización, capacidades de generación de código y soporte a múltiples plataformas y tecnologías juegan un papel preponderante. Visual WADE cubre todas ellas, y además proporciona un entorno altamente extensible, en el cual nuevos compiladores suponen la instanciación de un compilador genérico y la inclusión de la implementación resultante como un nuevo fichero Python que no

[28] De hecho, una experiencia reciente en SUMA demostró la viabilidad de desarrollar una aplicación de inventario para la empresa en tres horas, de las cuales más de dos fueron dedicadas al refinamiento de la apariencia visual de la interfaz.

requiere la recompilación de ningún módulo previamente existente.

El grado de automatismo y sincronización que Visual WADE proporciona al desarrollo de modelos en OO-H se unen a un potente entorno de prototipado, que permite la visualización de los resultados de las actividades de diseño en etapas tempranas del desarrollo. Este entorno incluye capacidades de navegación, que permiten validar las decisiones de diseño relativas tanto al contenido como a la navegación y presentación de la interfaz. Modificaciones en cualquiera de los modelos son propagadas de manera automática al resto de modelos, de manera que no existan inconsistencias. Todo ello produce aplicaciones web de una mayor calidad y que requieren un menor número de iteraciones.

Por último, un compilador de modelos permite la generación tanto de la interfaz como de los módulos de bases de datos, lógica y documentación que la acompañan, dando lugar a una arquitectura extensible donde la independencia entre las capas de presentación, lógica y persistencia está garantizada.

Visual WADE ha sido aplicado a diversos proyectos reales, y los resultados obtenidos nos hacen ser optimistas respecto a las posibilidades de implantación de métodos hipermediales que integran facilidades de generación de código a partir de modelos en el mundo empresarial.

10. Conclusiones y trabajos futuros

Desde el punto de vista tanto creativo como de negocio, sólo alrededor del 25% de Internet ha sido inventado.

JAKE WINEBAUM, CHAIRMAN, BUENA VISTA INTERNET GROUP
(FAST COMPANY)

10.1 Conclusiones finales

Hoy en día es imposible negar el impacto que Internet en general y la WWW en particular están teniendo sobre el modo de trabajar en muchas organizaciones. Este hecho ha provocado un número creciente de aplicaciones cada vez más complejas y dinámicas en términos de contenido, estructura, comportamiento e interfaz, que siguen sin embargo desarrollándose en su mayor parte de una manera artesanal.

La necesidad de dar respuesta a los nuevos retos que plantea este tipo de aplicaciones ha motivado el interés de la comunidad científica en el estudio e implantación de procesos de desarrollo sistemáticos que contribuyan a incrementar sus probabilidades de éxito de implantación.

Desde el Grupo de Ingeniería Web de la Universidad de Alicante[1], al cual pertenece la autora del presente trabajo, compartimos este interés, y es por ello que en esta tesis se presenta un método de desarrollo de interfaces hipermediales con acceso a funcionalidad conocido como OO-H, y que creemos que supone un paso adelante en este camino hacia la sistematización en el desarrollo de este tipo de aplicaciones.

[1] *http://www.dlsi.ua.es/iwad/*

OO-H es una aproximación dirigida por los requisitos de usuario, orientada a objetos y parcialmente basada en estándares, características que se han demostrado deseables para incrementar las posibilidades de implantación y uso de este tipo de métodos. Al igual que otros métodos, OO-H aboga por la separación explícita entre contenido, navegación y presentación, y define una trazabilidad entre modelos que permite la propagación de los cambios entre los distintos niveles de la especificación.

El proceso de desarrollo de una interfaz hipermedial en OO-H comienza con la fase de recogida de requisitos, en la que el diseñador introduce un Diagrama de Casos de Uso (compatible UML) que refleja los requisitos funcionales detectados durante su interacción con el cliente. A partir de este diagrama, una actividad de análisis de navegación permite la agrupación de requisitos siguiendo distintos criterios. Esta agrupación proporciona una estructura de navegación de primer nivel de la interfaz que es independiente no sólo de plataforma y tecnología, sino también de dispositivo de acceso. Por otro lado, una actividad de análisis y diseño de dominio permite la captura de un Diagrama de Clases (también compatible UML) que refleja la información de dominio y las interfaces ofertadas por los módulos de lógica subyacentes, si existiesen.

A partir de toda esta información el diseñador puede proceder al diseño de navegación de la aplicación mediante la creación de un Diagrama de Acceso Navegacional (DAN). Este concepto de navegación incluye en OO-H no sólo consultas a través del espacio de la información, sino que captura también aspectos relativos a las interfaces de servicio que pueden ser ofertadas al usuario mientras éste navega a través de dicha información. En este sentido, la semántica asociada a los distintos tipos de enlace en OO-H permite modelar las situaciones de interacción más comunes que podemos encontrar en aplicaciones web operativas.

Una vez diseñado el DAN, un proceso automático permite la generación de la estructura lógica de la aplicación, contenida en un Diagrama de Presentación Abstracta (DPA), que acerca el modelo de la interfaz a un modelo de implementación web basado en recursos (ficheros) y redistribuye toda la información capturada

en los modelos anteriores en una estructura de páginas abstractas, en función de los parámetros definidos sobre los enlaces en el modelo navegacional. Es precisamente sobre este concepto sobre el que el diseñador puede realizar una actividad de refinamiento a un menor nivel de abstracción, para lo cual OO-H proporciona un Catálogo de Patrones que, dotados de una regla de transformación, pueden ser aplicados directamente sobre los modelos, acelerando de este modo el proceso de desarrollo al mismo tiempo que asegura el tratamiento coherente del rasgo que se trate.

Además, las distintas páginas abstractas contenidas en el DPA son la base sobre la que se realiza la actividad de diseño de presentación. El motivo es que el concepto de página abstracta es fácilmente comprensible por personas no familiarizadas con la Ingeniería del Software, como puede ser el caso de los diseñadores gráficos. Para el desarrollo de esta actividad, OO-H proporciona un último tipo de diagrama, el Diagrama de Diseño Visual (DDV), que permite la manipulación individual de cada una de esas páginas de una manera todavía independiente de plataforma o tecnología de implementación final. Internamente, toda la información contenida tanto en el DPA como en el DDV se almacena en una taxonomía de plantillas XML que separan los distintos aspectos que intervienen en la definición de una interfaz y contienen toda la información necesaria para su posterior generación automática, independizando de este modo el proceso de generación del entorno de modelado específico de OO-H.

Por último, la especificación de la interfaz puede ser enriquecida con cuestiones relativas a políticas de personalización. En OO-H la integración de estas políticas se produce de manera natural, es decir, utilizando los mismos mecanismos que se utilizan para especificar cualquier otro tipo de restricciones en el sistema. Para su implementación OO-H propugna la construcción de un modelo de usuario, particular para cada aplicación. Además, OO-H proporciona un modelo de referencia extensible que, de partida, es capaz de recoger y analizar la información necesaria para dar soporte a algunas de las políticas más comunes en la actualidad.

OO-H acompaña esta propuesta con una herramienta CAWE denominada Visual WADE. Visual WADE incluye, entre otros rasgos, un potente entorno de prototipado de la especificación contenida en el DPA y los distintos DDV. Este entorno permite la validación del diseño por parte tanto del diseñador como del cliente de la aplicación, sin necesidad de generar ningún tipo de código. El resultado de esta validación puede ser utilizado para realizar una nueva iteración sobre el método, con el fin de ir refinando los distintos diagramas.

Una vez obtenida una especificación satisfactoria, Visual WADE incluye un conjunto de compiladores de modelos que, partiendo de la especificación textual de la interfaz, permiten la generación automática de código en la plataforma y tecnología deseada. En este momento es posible generar interfaces en tecnologías tan diversas como PHP, ASP y JSP.

De todo lo dicho hasta el momento, se pueden extraer como principales aportaciones de OO-H:

- Una entrada al método basada en estándares que permite, mediante un mecanismo de importación de modelos, complementar el trabajo de diseño realizado en otras plataformas con un modelo de interfaz desarrollado en OO-H.
- Una estructuración navegacional de la interfaz de alto nivel basada en la agrupación de requisitos de usuario, que permite aumentar la coherencia entre implementaciones generadas para distintos dispositivos de acceso.
- Un modelo de diseño navegacional multinivel que, dirigido por los requisitos de usuario y basado en los conceptos de dominio, permite definir no sólo los caminos de navegación del usuario a través de la aplicación sino también su modo de interactuar con módulos de lógica y/o bases de datos preexistentes.
- Una taxonomía de enlaces de navegación cuya flexibilidad y potencia semántica permite el modelado de nuevas posibilidades de interacción.

- Una taxonomía de modos de introducción de parámetros que permite modelar formularios de entrada no triviales de manera análoga a como se modelan el resto de caminos de navegación del sistema.
- Un modelo de referencia que, con la ayuda de un modelo de usuario, permite definir políticas de personalización de manera análoga a como se definen el resto de restricciones del sistema.
- Un modelo lógico de interfaz basado en el concepto de *página abstracta*, fácilmente comprensible por personas no técnicas como pueden ser los diseñadores gráficos de la interfaz.
- Un lenguaje de patrones cuya aplicación automática sobre la especificación textual de la interfaz acelera el proceso de desarrollo, facilita el tratamiento homogéneo de los distintos refinamientos y documenta prácticas de probada eficacia en el ámbito de las aplicaciones hipermediales.
- Un entorno de diseño de presentación intuitivo, totalmente integrado con el resto de modelos.
- Una taxonomía de plantillas XML que permite la especificación textual de las interfaces hipermediales y separa el tratamiento de contenido, navegación y distintos aspectos de presentación, facilitando así la evolución de las especificaciones.
- Un entorno de prototipado automático de la interfaz modelada que permite la validación de la aplicación tanto por parte del usuario como por parte del diseñador, incluso con versiones parciales de modelos.
- Un entorno de desarrollo que garantiza la sincronía entre modelos, preserva los cambios de nivel inferior ante modificaciones en modelos de un nivel de abstracción superior y facilita por tanto el desarrollo incremental de interfaces hipermediales.
- Un entorno de generación que incorpora un conjunto de compiladores de modelos para distintos lenguajes y plataformas.

10.2 Trabajos futuros

En general, desde OO-H consideramos que actualmente en el campo de la Ingeniería Web existen dos grandes áreas de trabajo abiertas:

- Mejora del grado de soporte dado en las distintas aproximaciones a aspectos avanzados del desarrollo de una aplicación, como son los requisitos no funcionales, los servicios, el flujo de trabajo, aspectos de seguridad y personalización, la integración de componentes, etc.
- Mejora de aspectos técnicos e ingenieriles en las distintas aproximaciones: mayor automatización de las actividades, aumento de la trazabilidad entre modelos, desarrollo de entornos que garantizen la sincronía entre los mismos, estandarización de notaciones y procesos, técnicas de verificación automática, inclusión de parámetros de calidad, etc.

En primer lugar, a pesar de que un gran número de conceptos de la Ingeniería Web han alcanzado un considerable grado de madurez, existen otros sobre los que aún no existe un acuerdo generalizado, lo que invariablemente dota de un grado elevado de inestabilidad a los métodos, procesos y notaciones que pretenden su abstracción. Así cuestiones como ¿es necesario un modelo de tareas? o ¿cómo se define un modelo de diálogo usuario-sistema, y a qué nivel de detalle debe llegar? ¿Es necesario un modelo de navegación independiente de dispositivo? ¿Qué papel juegan los componentes software, y en qué fase del proceso de autoría deberían ser considerados? siguen causando controversia.

Por tanto, creemos que es fundamental que la comunidad hipermedial siga trabajando en el análisis e inclusión de estas cuestiones en el seno de las distintas propuestas, con el fin de proporcionar a las organizaciones un conjunto de guías claras que las ayuden en el proceso de desarrollo de aplicaciones. En OO-H estos esfuerzos de avance se dirigen actualmente hacia (1) una evolución del diseño de la navegación, que actualmente se intenta abordar desde

un punto de vista más cercano a la ingeniería de requisitos, y (2) un incremento del grado integración de servicios no implementados como operaciones CRUD. Asimismo, cuestiones relativas a la externalización de políticas de personalización, la integración del flujo de trabajo y el modelado de aspectos de seguridad de la aplicación, que han sido tratadas sólo de manera tangencial en la mayoría de las propuestas, están actualmente siendo estudiados en el seno de OO-H.

En cuanto a la segunda área de trabajo, pensamos que, de cara a la implantación de los distintos métodos hipermediales en el ámbito empresarial, se hace necesaria una mejora en determinados aspectos técnicos e ingenieriles, entre los que destacamos una mayor automatización de las actividades, un aumento de la trazabilidad entre modelos, el desarrollo de entornos que garantizen la sincronía entre los mismos, la estandarización de notaciones y procesos, la inclusión de técnicas formales de verificación automática o la inclusión de parámetros de calidad que permitan medir la bondad de los modelos y, por tanto, de las aplicaciones implementadas a partir de estos modelos. El entorno de desarrollo de OO-H cubre algunos de estos requisitos (en concreto un grado elevado de automatización de actividades, la trazabilidad de modelos y una garantía de sincronía entre los mismos). Por tanto, quedan planteados como trabajos futuros los relacionados con la estandarización de modelos (mediante la definición de un *perfil*[2] UML) y la inclusión tanto de técnicas de verificación automática como de métricas de calidad sobre dichos modelos.

En resumen, las principales líneas de trabajo futuro propuestas en el seno de OO-H son:

- Estandarización de la propuesta, mediante la creación de un *perfil* UML.
- Estandarización del proceso de desarrollo de interfaces mediante OO-H.

[2] profile

- Aumento del peso que los casos de uso tienen sobre la especificación de la estructura de navegación de la interfaz de la aplicación.
- Inclusión de modelos de tareas (análisis de flujo de la aplicación) que enriquezcan la semántica de la especificación y mejoren la calidad de uso de las aplicaciones.
- Profundización en los aspectos relacionados con la personalización (externalización de las políticas) y la seguridad en las aplicaciones.
- Profundización en los aspectos de interconexión con módulos de lógica y/o bases de datos externas. Integración completa de los Servicios Web.
- Ampliación del Catálogo de Patrones de OO-H para dar cabida a nuevas posibilidades de implementación y a nuevas soluciones a problemas comunes en el ámbito de las interfaces hipermediales.
- Ampliación de la propuesta para permitir el modelado de distintas arquitecturas de aplicación.
- Inclusión de nuevos generadores que den soporte a nuevas plataformas y tecnologías.
- Generación de datos de prueba que enriquezcan el prototipado de la aplicación.
- Generación automática de conjuntos de pruebas que permitan evaluar en un entorno controlado la corrección de la aplicación final generada.
- Integración de métricas en el entorno de desarrollo que permitan evaluar de manera automática la bondad de los modelos.
- Formalización de los distintos modelos para facilitar su verificación automática.

Para finalizar, nos gustaría destacar cómo estas líneas de investigación se están viendo en muchos casos potenciadas por (1) el creciente esfuerzo de los distintos grupos de investigación para

promover la aplicación de las distintas propuestas a nivel empresarial y (2) la cada vez mayor interacción entre distintos grupos de investigación.

Este esfuerzo, que ya se está viendo materializado en proyectos tanto a nivel nacional como internacional[3], está influyendo de manera decisiva en la rápida evolución de las distintas propuestas. De hecho la mayoría de líneas de investigación propuestas (inclusión de aspectos relativos a la seguridad, análisis de flujo de tareas, inclusión de técnicas y tecnologías en un principio ajenas a la web, como es el caso de los patrones de diseño o los agentes software, el soporte a la verificación de los modelos, la implantación de métricas de calidad de las interfaces, etc.) son propuestas de trabajo que en OO-H han surgido como resultado tanto de la experienca de implantación del método para el desarrollo de aplicaciones reales como por esta sinergía entre disciplinas y aproximaciones, que se benefician del conocimiento y experiencia acumulada del resto.

10.3 Publicaciones realizadas

Los resultados presentados en este trabajo han sido contrastados en un artículo en una revista internacional, un capítulo de libro, siete artículos en congresos internacionales, cuatro artículos en congresos nacionales, una ponencia invitada y una demostración. Hemos agrupado los artículos por tipo de publicación, y dentro de ellos los hemos clasificado por año de publicación:

- **Artículos en Revistas Internacionales**
  - J. Gómez, C. Cachero and O. Pastor. "On Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach". **IEEE Multimedia** 8(2), pa-

[3] En el caso de OO-H, algunos ejemplos son el convenio entre SUMA y OO-H, comentado en el capítulo 9 o la inclusión del grupo de Ingeniería Web de la Universidad de Alicante en una propuesta de Red Europea de Excelencia en Ingeniería Web (EWENE, *European Web Engineering Network of Excellence*)

ges 20-32. Special Issue on Web Engineering. April 2001. Impact Factor JCR: 0.507. Ranking 36/75 in category *Computer Science/Sofware Engineering.*

- **Capítulos de libro**
  - J. Gómez and C. Cachero. "OO-H: Extending UML to Model Web Interfaces". Information Modeling for Internet Applications. Idea Group Publishing. 2002.
- **Artículos en Congresos Internacionales**
  - J. Gómez, C. Cachero and O. Pastor. "Extending a Conceptual Modelling Approach to Web Application Design". 12^{th} International Conference on Advanced Information Systems (**CAiSE'00**). Stockholm. June 2000. LNCS 1789, pages 79-93. Springer-Verlag. Impact Factor JCR: 0.390
  - C. Cachero, J. Gómez and O. Pastor. "OO Conceptual Modeling of Web Application Interfaces: the OO-HMethod Abstract Presentation Model". 1^{st} International Conference on Electronic Commerce and Web Technologies (**EC-Web'00**). Greenwich. September 2000. LNCS 1875, pages 206-215. Springer-Verlag. Impact Factor JCR: 0.390
  - C. Cachero, J. Gómez and O. Pastor. "Conceptual Design of Electronic Product Catalogs Using Object-Oriented Hypermedia Modeling Techniques". 1^{st} Workshop on Conceptual Modeling Approaches for E-Business (**eCOMO'00**). Salt Lake City. October 2000. LNCS 1921, pages 19-30. Springer-Verlag. Impact Factor JCR: 0.390
  - J. Gómez, C. Cachero and O. Pastor. "OO-HMethod: un método de diseño de lugares web". Terceras Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes Software (**IDEAS'00**). Cancún, México. Abril 2000. Actas de la Conferencia, pages 133-144.
  - C. Cachero, J. Gómez, A. Párraga and O. Pastor. "Conference Review System: a case of study". 1^{st} International Workshop

on Web Oriented Software Technology (**IWWOST'01**). Valencia. June 2001. Workshop Proceedings, pages 195-227.

- C. Cachero, I. Garrigós and J. Gómez. "Personalización de Aplicaciones en OO-H". Quintas Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes Software (**IDEAS'02**). La Habana, Cuba. Abril 2002. Actas de la Conferencia.
- C. Cachero, N. Koch, J. Gómez and O. Pastor. "Conceptual Navigation Analysis: a device and platform independent navigation specification". 2^{nd} International Workshop on Web Oriented Software Technology (**IWWOST'02**). Málaga, Spain. June 2002. Workshop Proceedings.

- **Artículos en congresos nacionales**
  - C. Cachero, J. Gómez y O. Pastor. "Modelando aspectos de navegación y presentación en aplicaciones hipermediales". V Jornadas de Ingeniería del Software y Bases de Datos (**JISBD'00**). Valladolid. Noviembre 2000. Actas de la Conferencia.
  - C. Cachero, J. Gómez, A. Párraga y O. Pastor. "Extending UML for the migration of Legacy Systems to the Web". VI Jornadas de Ingeniería del Software y Bases de Datos (**JISBD'01**). Almagro. Noviembre 2001. Actas de la Conferencia.
  - C. Cachero and J. Gómez. "Advanced Conceptual Modeling of Web Applications: Embedding Operation Interfaces in Navigation Design". VII Jornadas de Ingeniería del Software y Bases de Datos (**JISBD'02**). El Escorial, Madrid. Noviembre 2002. Aceptado.
  - Irene Garrigós, Cristina Cachero y Jaime Gómez. "Modelado Conceptual de aplicaciones adaptivas y proactivas en OO-H". II Taller de Ingeniería del Software Orientada al Web (**WebE'2002**). En las VII Jornadas de Ingeniería del Software y Bases de Datos (**JISBD'02**). El Escorial, Madrid. Noviembre 2002. Aceptado.

Volver al Índice/Tornar a l'índex

- **Ponencias Invitadas**
  - J. Gómez, C. Cachero y O. Pastor. "Modelado conceptual de aplicaciones web independientes de dispositivo". Ponencia invitada en las VI Jornadas de Ingeniería del Software y Bases de Datos (**JISBD'01**). Almagro, España. Noviembre 2001. Actas de la Conferencia.
- **Demostraciones**
  - J. Gómez, C. Cachero and A. Párraga. "LES Objects: A Model-based Code Generation Environment for Object-Oriented Conceptual Modeling of Web Application Interfaces". Demonstration at the 16^{th} European Conference on Object-Oriented Programming (**ECOOP'02**). Málaga. Spain. June 2002. ECOOP'02 Demonstration Booklet.

Agradecimientos

Desde este trabajo quiero mostrar mis más sincero reconocimiento al equipo desarrollador de VisualWADE, cuyo excepcional trabajo ha hecho posible la materialización de las ideas presentadas en esta tesis. Este equipo está formado por:

- *Antonio Párraga (Arquitecto Software)*
- *Óscar Asensi (Ingeniero Software)*
- *David de Francisco (Ingeniera Software)*
- *Irene Garrigós (Ingeniera Software)*
- *Luis Lucas (Ingeniero Software)*
- *Susana Vicente (Ingeniera Software)*
- *Jaime Gómez (Director del Proyecto Investigador UA-Suma Gestión Tributaria)*

Referencias

ACM (ed). 2000 (08). *Special issue on Personalization.* Vol. 43.

Alexander, C. 1979. *The timeless Way of Building.* Oxford University Press.

ArgoUML. 2002. *ArgoUML Project Home.* http://argouml.tigris.org.

Atzeni, P., & Parente, A. 2001. Specification of Web applications with ADM-2. *In:* (Pastor, 2001).

Atzeni, P., Mecca, G., & Merialdo, P. 1997 (08). To Weave the Web. *Pages 206–215 of: Proceedings of VLDB'97 International Conference.*

Atzeni, P., Mecca, G., & Merialdo, P. 1998. Design and Maintenance of Data-Intensive Web Sites. *In:* (Schek *et al.*, 1998).

Avery, C., & Zeckhauser, R. 1997. Recommender Systems for Evaluating Computer Messages. *Communications of the ACM*, 03.

Avison, D., Lan, F., Myers, M., & Nielsen, A. 1999. Action Research. *Communications of the ACM*, **42**(1), 94–97.

Baresi, L., Garzotto, F., & Paolini, P. 2001. Extending UML for Modeling Web Applications. *Proceedings of the 34th International Conference on System Sciences*, 01.

Barry, C., & Lang, M. 2001. A Survey of Multimedia and Web Development Techniques and Methodology Usage. *IEEE Multimedia Special Issue on Web Engineering*, 04, 52–60.

Bell, R. 1998. Code Generation from Object Models. *Embedded Systems Programming*, 1–9.

Bernstein, M. 1998. Patterns of Hypertext. *Pages 21–29 of: ACM Conference on Hypertext and Hypermedia.*

Bichler, M., & Nusser, S. 1996. Developing Structured WWW-Sites with W3DT. *In: Proceedings of WebNet'96 International Conference.* AACE.

Bieber, M. 2000. Hypermedia: A Design Philosophy. *ACM Computing Surveys*, 12.

Bieber, M., Oinas-Kukkonen, H., & Balasubramanian, V. 1999. Hypertext Functionality. *ACM Computing Surveys*, **31**(4).

Billard, D. 1998. Transactional Services for the Internet. *In: Proceedings of WebDB'98 International Conference.* ACM.

Boehm, B. 1976. Software Engineering. *IEEE Transactions on Computers*, 12, 1226–1241.

Booch, G., Rumbaugh, J., & Jacobson, I. 1999. *The Unified Modeling Language user guide.* Addison Wesley Longman Publishing Co.

Bra, P. De. 1999. Design Issues in Adaptive Web-Site Development. *In: Proceedings of IWASUM'99 International Workshop.*

Bra, P. De, Houben, G.J., & Wu, H. 1999. AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. *Pages 147–156 of: ACM Conference on Hypertext and Hypermedia.* ACM.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. 1996. *A System of Patterns.* Wiley.

Butler, T. 2000. Transforming information systems development through computer-aided systems engineering (CASE): lessons from practice. *Information Systems Journal*, **10**(10), 167–193.

Cachero, C. 1999 (12). *Memoria de Investigación 1999.* Tech. rept. Universidad de Alicante.

Cachero, C. 2000 (12). *Memoria de Investigación 2000.* Tech. rept. Universidad de Alicante.

Cachero, C. 2001 (10). *Memoria de Investigación 2001.* Tech. rept. Universidad de Alicante.

Cachero, C., & Gómez, J. 2002. Advanced Conceptual Modeling of Web Applications: Embedding Operation Interfaces in Navigation Design (Not yet published). *In: VII Jornadas de Ingeniería del Software y Bases de Datos.* JISBD Workshop Proceedings.

Cachero, C., & Koch, N. 2002. *Navigation Analysis and Navigation Design in OO-H and UWE.* Tech. rept. University of Alicante and München Universität.

Cachero, C., Gomez, J., & Pastor, O. 2000a. Conceptual Design of Electronic Product Catalogs using Object-Oriented Hypermedia Modeling Techniques. *Pages 19–30 of: 1st Workshop on Conceptual Modeling Approaches for E-Business (ECOMO).* Springer-Verlag. Lecture Notes in Computer Science.

Cachero, C., Gómez, J., & Pastor, O. 2000b. Modelando Aspectos de Navegación y Presentación en Aplicaciones Hipermediales. *In: V Jornadas de Ingeniería del Software y Bases de Datos.* JISBD Workshop Proceedings.

Cachero, C., Gómez, J., & Pastor, O. 2000c. Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO-HMethod Presentation Abstract Model. *In: EC-Web 2000.* 1^{st} *International Conference on Electronic Commerce and Web Technologies.* Springer-Verlag. Lecture Notes in Computer Science.

Cachero, C., Gomez, J., & Pastor, O. 2000d. OO-HMethod: Un Método de Diseño de Lugares Web. *Pages 133–144 of: II Workshop Iberoamericano de Ingeniería del Software y Bases de Datos IDEAS'02.* Actas del Congreso.

Cachero, C., Gómez, J., Párraga, A., & Pastor, O. 2001a. Conference Review System: A Case of Study. *In:* (Pastor, 2001).

Cachero, C., Gómez, J., Parraga, A., & Pastor, O. 2001b. Extending UML for the migration of Legacy Systems to the Web. *In: VI Jornadas de Ingeniería del Software y Bases de Datos.* JISBD Workshop Proceedings.

Cachero, C., Koch, N., Gómez, J., & Pastor, O. 2002a. Conceptual Navigation Analysis: a device and platform independent navigation specification. *In:* (Schwabe *et al.*, 2002).

Cachero, C., Koch, N., Gómez, J., & Pastor, O. 2002b. *Navigation analysis vs. navigation design.* Tech. rept. University of Alicante and München Universität and Valencia University of Technology.

Cachero, C., Garrigós, I., & Gómez, J. 2002c. Personalización de Aplicaciones en OO-H. *In: IV Workshop Iberoamericano de Ingeniería del Software y Bases de Datos IDEAS'02.* Actas del Congreso.

Carroll, J., & Aaronson, A. 1988 (09). Learning by Doing With Simulated Intelligent Help. *In: Communications of the ACM.*

Casteleyn, S., & Troyer, O. De. 2002. Exploiting Link Types during the Web Site Design Process to Enhance Usability of Web Sites. *In:* (Schwabe *et al.*, 2002).

Ceri, S., Fraternali, P., & Paraboschi, S. 1999 (02). Data-Driven One-to-One Web Site Generation for Data-Intensive Applications. *In: Proceedings of VLDB'99 International Conference.*

Ceri, S., Fraternali, P., & Bongio, A. 2000. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, **33**(1-6), 137–157.

Conallen, J. 1999. *Building Web Applications with UML.* Addison Wesley Longman.

Customer Profile EXchange. 2001. *Customer Profile EXchange.* www.cpexchange.org.

Dart, S. 1999. Containing de Web Crisis Using Configuration Management. *In: First ISCE Workshop on Web Engineering.* ACM.

Díaz, A., Isakowitz, T., Maiorana, V., & Gilabert, G. 1995. RMC: a Tool to design WWW Applications. *In: 5th International World Wide Web Conference.*

DCOM. 2002. *DCOM.* www.microsoft.com.

DSML. 2000. *Directory Services Markup Language.* http://www.dsml.org/.

Enguix, C.F., & Davis, J.G. 1999. Filling the Gap: New Models for Systematic Page-based Web Application Development and Maintenance. *In: 8th WWW International Workshop on Web Engineering.*

Evitts, P. 2000. *A UML Pattern Language.* Software Engineering Series.

Fayad, E. M., & Schmidt, D. C. 1997. Object-oriented application frameworks. *CACM: Communications of the ACM.*, **40**(10), 32–38.

Fernández, F. M., Florescu, D., Kang, J., Levy, A., & Suciu, D. 1998 (10). Catching the Boat with Strudel: Experiences with a Web-Site Management System. *Pages 414–425 of: Proceedings of ACM SIGMOD International conference on Management of data.*

Fernández, M., Suciu, D., & Tatarinov, I. 1999 (05). Declarative Specification of Data-intensive Web sites. *In: USENIX conference on Domain Specific Languages.*

Florescu, D., Levy, A., & Mendelzon, A. 1998 (09). Database Techniques for the World Wide Web: a Survey. *Pages 59–74 of: Sigmod Record*, vol. 27.

Fraternali, P. 1999. Tools and Approaches for Developing Data-Intensive Web Applications: a Survey. *ACM Computing Surveys*, 09.

Fraternali, P., & Paolini, P. 1998. A Conceptual Model and a Tool Environment for Developing more Scalable, Dynamic, and Customizable Web Applications. *In:* (Schek *et al.*, 1998).

Fraternali, P., & Paolini, P. 2000. Model-driven development of Web applications: the Autoweb System. *ACM Transactions on Information Systems (TOIS)*, **18**(4), 323–382.

Fraternali, P., Paolini, P., Fedon, G., & Oggioni, M. 1998. *Autoweb System.* www.ing.unico.it/autoweb/.

Fraternali, P., Matera, M., & Maurino, A. 2002. WQA: an XSL Framework for Analyzing the Quality of Web Applications. *In:* (Schwabe *et al.*, 2002).

French, W.L., & Bell, C.H. 1996. *Desarrollo Organizacional.* 5 edn. Prentice-Hall.

Gaedke, M., & Rehse, J. 2000. Supporting Compositional Reuse in Component-Based Web Engineering. *Pages 927–933 of: SAC (2)*.

Gaedke, M., Lyardet, F., & Werner-Gellersen, H. 1999. Hypermedia Patterns and Components for Building better Web Information Systems. *In: ACM Conference on Hypertext and Hypermedia*.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 1995. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley.

Garrigós, I., Cachero, C., & Gómez, J. 2002. Modelado conceptual de aplicaciones adaptivas y proactivas en OO-H. *In: VII Jornadas de Ingeniería del Software y Bases de Datos*. JISBD Workshop Proceedings.

Gartner Group, Inc. 2001 (02). *The Future of Web Services: Dynamic Business Webs. Market Analysis*.

Garzotto, F., Paolini, P., & Schwabe, D. 1991. HDM-a model for the design of hypertext applications. *Pages 313–328 of: ACM Conference on Hypertext*. ACM Press.

Garzotto, F., Paolini, P., & Schwabe, D. 1993. HDM A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems (TOIS)*, **11**(1), 1–26.

Gellersen, H., & Gaedke, M. 1999. Object-Oriented Web Application Development. *IEEE Internet Computing*, January, 60–68.

Gellersen, H. W., Wicke, R., & Gaedke, M. 1997. WebCompostion: an object-oriented support system for the Web Engineering Lifecycle. *Computer Networks and ISDN Systems. Special Issue on the 6 th Intl. World-Wide Web Conference*, 1429–1437.

Genova Project. 2002. *Genova Project. http : //www.genera.no/2052/tilkunde/09.04/data/pages/page_display.asp?doc_id =* 93.

Germán, D.M., & Cowan, D.D. 2000. Towards a unified catalog of hypermedia design patterns. *In: 33rd Hawaii International Conference on System Sciences*.

Ginige, A. 1998 (12). Web Engineering: Methodologies for Developing Large and Maintainable Web-Based Information Systems. *Pages 89–92 of: IEEE International Conference on Networking*.

Ginige, A., & Murugesan, S. 2001. Web Engineering: an Introduction. *IEEE Multimedia Special Issue on Web Engineering*, 04, 14–18.

Gómez, J., & Cachero, C. 2002. *Information Modeling for Internet Applications*. Idea Group Publishing. Chap. OO-H: Extending UML to Model Web Interfaces, pages 144–173.

Gómez, J., Cachero, C., & Pastor, O. 2000. Extending a Conceptual Modelling Approach to Web Application Design. *Pages 79–93 of:* 12^{th} *International Conference on Advanced Information Systems (CAiSE'00)*, vol. 1789. Springer-Verlag. Lecture Notes in Computer Science.

Gómez, J., Cachero, C., & Pastor, O. 2001a. Conceptual Modelling of Device-Independent Web Applications. *IEEE Multimedia Special Issue on Web Engineering*, 04, 26–39.

Gómez, J., Cachero, C., & Pastor, O. 2001b. Modelado conceptual de aplicaciones web independientes de dispositivo. Ponencia invitada. *In: VI Jornadas de Ingeniería del Software y Bases de Datos.* JISBD Workshop Proceedings.

Gómez, J., Cachero, C., & Párraga, A. 2002. *LES Objects: A Model-based Code Generation Environment for Object-Oriented Conceptual Modeling of Web Application Interfaces.* ECOOP'02 Booklet. http://ecoop2002.lcc.uma.es/tpDemonstrations.htm.

Goeschka, K. M., & Schranz, M. W. 2001. Client and Legacy Integration in Object-Oriented Web Engineering. *IEEE Multimedia Special Issue on Web Engineering*, 04, 32–41.

Gómez, J. 2000 (Febrero). *Generación Automática de Componentes Software a partir de Modelos Conceptuales Orientados a Objetos.* Ph.D. thesis, Departamento de Lenguajes y Sistemas Informáticos. Universidad de Alicante.

Halasz, F., & Schwartz, M. 1994. The Dexter Hypertext Reference Model. *Communications of the ACM*, **37**(2), 30–39.

Hardman, L., Bulterman, C., & van Rossum, G. 1994. The Amsterdam Hypermedia Model. *Communications of the ACM*, **37**(2), 50–62.

Herzum, P. 2001 (12). *Distributed Enterprise Architecture Advisory Service.* Tech. rept. Cutter Consortium.

Isakowitz, T., Stohr, E. A., & Balasubramanian, V. 1995. RMM: A Methodology for Structured Hypermedia Design. *CACM: Communications of the ACM.*, 08, 34–44.

Jacobson, I., Booch, G., & Rumbaugh, J. 1999. *The Unified Software Development Process.* Addison Wesley.

Jayaratna, N. 1990. Systems Analysis: the need for a better understanding. *International Journal of Information Systems Management*, 228–234.

Kappel, G., Retschitzegger, W., & Schwinger, W. 2000 (11). Modeling Customizable Web Applications - A Requirement's Perspective. *In: 2000 Kioto International Conference on Digital Libraries.*

Kesseler, M. 1995. A schema-based approach to HTML authoring. *In: 4th International Conference on the WWW.*

Kirda, E., Jazayeri, M., Kerer, C., & Schranz, M. 2001. Experiences in Engineering Flexible Web Services. *IEEE Multimedia Special Issue on Web Engineering*, 01, 58–65.

Koch, N. 2000 (01). *Hypermedia Systems Development based on the Unified Process.* Tech. rept. Ludwig-Maximilians-Universität Munchen.

Koch, N. 2001. *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process.* Ph.D. thesis, Ludwig-Maximilians-Universität München.

Koch, N., & Wirsing, M. 2001 (07). Software Engineering for Adaptive Hypermedia Applications. *In: 3rd Workshop on Adaptive Hypertext and Hypermedia (8th International Conference on User Modeling).*

Koch, N., Kraus, A., & Hennicker, R. 2001 (05). The Authoring Process of the UML-based Web Engineering Approach. *In: Proceedings of the 1st International Workshop on Web-Oriented Software Technology.*

Kramer, J., Noronha, S., & Vergo, J. 2000 (08). A User-Centered Design Approach to Personalization. *In: ACM Computing Surveys.*

Kraus, A., & Koch, N. 2002 (06). Generation of web applications from UML models using an XML publishing framework. *In: 6th World Conference on Integrated Design and Process Technology (IDPT).*

L.A.Maciaszek. 2001. *Requirements Analysis and System Design. Developing Information Systems with UML.* Addison Wesley.

Lange, D. 1996. An Object-Oriented Design Approach for Developing Hypermedia Information Systems. *Journal of Organizational Computing and Electronic Commerce*, **3**(6), 269–293.

Lange, D. B. 1993 (01). Object-oriented hypermodeling of hypermedia-supported information systems. *Pages 380–389 of: 26th Hawaii International Conference on System Sciences.*

Lee, H., Lee, C., & Yoo, C. 1998. A Scenario-Based Object Oriented Methodology for Developing Hypermedia Information Systems. *In: 31st Hawaii International Conference on System Sciences (HICSS'98).* IEEE Computer Society.

Lee, S. C. 1998. IDM: a methodology for Intranet Design. *Pages 51–67 of: International Conference on Information Systems.* ACM.

Liddle, S. 2001. Paper Review System Overview. *Pages 153–165 of: 1st International Workshop on Web Oriented Software Technology (IWWOST'01).* Workshop Proceedings.

Lowe, D. B., & Hall, W. 1999. *Hypermedia and the Web: an Engineering Approach.* John Wiley & Sons.

Lowe, D. B., & Henderson-Sellers, B. 2001. Web Development: Addressing Process Differences. *Cutter IT Journal.*

Lowe, D. B., Bucknell, J. A., & Webby, G. R. 1999. Improving Hypermedia Development: A Reference Model-Based Process Assessment Method. *Proceedings of the 10th ACM Conference on Hypertext and hypermedia: returning to our diverse roots.*, 139–146.

LUA. 2002. *The LUA programming language.* http://www.lua.org/.

Manola, F. 1999. Technologies for a Web Object Model. *IEEE Internet Computing*, 01, 38–47.

McClure, S. 1998. *Web Application Development Developer Perspectives.* Tech. rept. IDC.

McIlhagga, M., Light, A., & Wakeman, I. 1998 (10). Towards a Design Methodology for Adaptive Applications. *In: ACM/IEEE International Conference on Mobile Computing and Networking.*

Mecca, G., Merialdo, P., & Atzeni, P. 1999a. Araneus in the era of XML. *IEEE Data Engineering Bulletin*, **22**(3), 19–26.

Mecca, G., Merialdo, P., Atzeni, P., & Crescenzi, V. 1999b (03). *The ARANEUS Guide to Web-Site Development.* Tech. rept. Universidad de Roma.

Mobasher, B., Cooley, R., & Srivastava, J. 1999 (11). Creating Adaptive Web Sites Through Usage-Based Clustering of URL's. *In: IEEE Workshop on Knowledge and Data Engineering Exchange.*

Molina, P. 1998. *Especificación de la Interfaz de Usuario en OO-Method.* Tech. rept. Universidad Politécnica de Valencia.

Molina, P., Meliá, S., & Pastor, O. 2002. User Interface Conceptual Patterns. *In: 9th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS 2002).* Springer Verlag.

Murugesan, S., Deshpande, Y., Hansen, S., & Ginige, A. 1999 (05). Web Engineering: A New Discipline for Development of Web-based Systems. *In: First ICSE Workshop on Web Engineering, International Conference on Software Engineering.*

MyXML. 2000. *MyXML Specification.* http://www.infosys.tuwien.ac.at/myxml/.

Nanard, J., & Nanard, M. 1995. Hypertext Design Environments and the Hypertext Design Process. *Communications of the ACM*, **8**(38), 49–56.

Nanard, J., & Nanard, M. 1999. Towards an Hypermedia Design Pattern Space. *In: 2nd Workshop in Hypermedia Development: Design Patterns in Hypermedia.*

Nanard, M., Nanard, J., & Kahn, P. 1998. Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. *Pages 11–20 of: HYPERTEXT '98. Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space—structure in hypermedia systems.*

Nielsen, J., Rosenfeld, L., Tognazzini, B., Rhodes, J., & Sullivan, T. 2001. *Usability Resources.* http://www.usableweb.com/.

Nunes, N. Jardim, & e Cunya, J. Falcao. 2000 (04). *Wisdom - Whitewater Interactive System Development with Object Models.* Tech. rept. Universidade da Madeira and Universidade do Porto.

Objectmatter. 2001. *Object Relational Mapping Strategies.* http://www.objectmatter.com/vbsf/docs/maptool/ormapping.html.

Olsina, L. 1998. Building a Web-based Information System Applying the Hypermedia Flexible Process Modeling Strategy. *In: 1st International Workshop on Hypermedia Development (Hypertext 98').*

Olsina, L., Godoy, D., Lafuente, G.J., & Rossi, G. 2001. Specifying Quality Characteristics and Attributes for Websites. *IEEE Multimedia.*

OMG. 2002. *Object Management Group.* www.omg.org.

Padak, N., & Padak, G. 1994. *Guidelines for Planning Action Research Projects.* http://archon.educ.kent.edu/Oasis/Pubs/0200-08.html.

Pang, R. 1998. *Data Conversion from Object-Oriented to Relational database and its Verification by use of Information Capacity.* www.cse.cuhk.edu.hk/ acm-hk/activity/pg/cityu-ptpang.pdf.

Paolini, P., & Garzotto, F. 1999. Design Patterns for the WWW hypermedia: problems and proposals. *In: 2nd Workshop in Hypermedia Development: Design Patterns in Hypermedia.*

Pastor, O. (ed). 2001. *International Workshop on Web-Oriented Software Technology.* Valencia University of Technology.

Pastor, O., Gómez, J., Insfrán, E., & Pelechano, V. 2001. The OO-Method Approach for Information Systems Modeling: From Object-Oriented Conceptual Modeling to Automated Programming. *Information Systems*, **26**(7), 507–534.

Paterno, F. 1999. *Model-based Design and Evaluation of Interactive Applications.* Springer Verlag, UK.

Paterno, F., & Mancini, C. 1999 (05). Developing Task Models from Informal Scenarios. *In: ACM Proceedings of CHI'99 conference on Human factors in computing systems.*

Pavón, J. 2001. *Personalización de servicios en la Web.* ZOCO 2001. http://tdg.lsi.us.es/ zoco/res/ppt/pavon.zip.

Personalization Consortium. 2002. *Personalization Consortium.* www.personalization.org.

PHP. 2002. *PHP: Hypertext Preprocessor.* http://www.php.net.

Párraga, A., Gómez, J., & Cachero, C. 2002. *Persistencia de Objetos en un modelo relacional para su uso en un ambiente Internet/Intranet.* Tech. rept. Universidad de Alicante.

Python. 2002. *Python Web Site.* http://www.python.org/.

Rational Rose. 2002. *Rational Rose Case Tool.* http://www.rational.com/.

RDF. 2002. *The Resource Description Framework.* http://www.w3.org/RDF/.

Retschitzegger, W., & Schwinger, W. 2000 (08). Towards Modeling of DataWeb Applications - A Requirement's Perspective. *Pages 149–155 of: Proceedings of the American Conference on Information Systems AMCIS 2000*, vol. 1.

Robie, J., & Bartels, D. 1994. *A comparison between Relational and OO Databases for OO Application Development.* White Paper. POET Software Corporation.

Rodríguez, J., & Díaz, O. 2002. Moving Service Dependencies at the Presentation Layer. *In: International Conference on Software Engineering (ICSE'02) (To be published).* ACM.

Rosenberg, D., & Scott, K. 1998. *Use Case Driven Object Modeling with UML. A Practical Approach.* Addison Wesley.

Rossi, G., Schwabe, D., & Garrido, A. 1997. Design Reuse in Hypermedia Applications Development. *Pages 57–66 of: Proceedings of the eight ACM conference on HYPERTEXT '97.*

Rossi, G., Schwabe, D., & Lyardet, F. 1999. Improving web information systems with navigational patterns. *In: 8th International World Wide Web Conference.* Elsevier.

Rossi, G., Schwabe, D., & Guimaraes, R. 2001 (05). Designing Personalized Web Applications. *Pages 275–284 of: Tenth International World Wide Web Conference.*

R.S. Pressman. 2000. *Software Engineering. A practitioner's approach (Fifth edition).* Mc Graw Hill.

RUP. 2001. *Rational Unified Process.* http://www.rational.com/products/rup/index.jsp.

S. M. Vivekanandan. 2002. *Asynchronous Linking in a ServiceOriented Architecture.* http://www.ecs.soton.ac.uk/ dem/workshops/ohs2002/positions/Vivekanandan.pdf.

Schek, Hans-Jörg, Saltor, Fèlix, Ramos, Isidro, & Alonso, Gustavo (eds). 1998. *Advances in Database Technology.* Lecture Notes in Computer Science, vol. 1377. Springer-Verlag.

Schwabe, D., & Almeida, R. 1998 (08). *OOHDM-WEB: Rapid Prototyping of Hypermedia Applications in the WWW.* Tech. rept. Dept.of Informatics. PUC-Rio.

Schwabe, D., & Almeida, R. 1999. A Method-based Web Application Development Environment. *In: Position Paper, Web Engineering Workshop, WWW8.*

Schwabe, D., Rossi, G., & Barbosa, D. J. 1996. Systematic Hypermedia Application Design with OOHDM. *Page 166 of: Proceedings of the the seventh ACM conference on HYPERTEXT '96.*

Schwabe, D., Almeida, R., & Moura, I. 1999a. Leveraging Template-based Website Implementations Using Design Methods. *In: 8th International World Wide Web Conference.*

Schwabe, D., Pontes, R., & Moura, I. 1999b. *OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW.*

Schwabe, D., Esmeraldo, L., Rossi, G., & Lyardet, F. 2001. Engineering Web Applications for Reuse. *IEEE Multimedia. Special Issue on Web Engineering,* 01-03, 20–31.

Schwabe, D., Pastor, O., Olsina, L., & Rossi, G. (eds). 2002. *International Workshop on Web-Oriented Software Technology.* ECOOP.

SOAP. 2001. *Simple Object Access Protocol.* http://www.develop.com/soap/.

Svensson, M., Höök, K., Laaksolahti, J., & Waern, A. 2001 (03). Social Navigation of Food Recipes. *In: SIGCHI'01.*

Takahashi, K., & Ling, E. 1997. Analysis and Design of Web-based Information Systems. *Pages 377–389 of: Proc. of Sixth International World Wide Web Conf.* Springer-Verlag. Lecture Notes in Computer Science.

Tauscher, L., & Greenberg, S. 1997. Revisitation patterns in World Wide Web navigation. *Pages 399–406 of: CHI '97. Proceeding of the CHI 97 conference on Human factors in computing systems.*

Thomson, J., Greer, J., & Cooke, J. 1998. Algorithmically Detectable Design Patterns for Hypermedia Collections. *In: 1st International Workshop on Hypermedia Development (Hypertext 98').*

Tochtermann, K., & Dittrich, G. 1996. The Dortmund Family of Hypermedia Models. *Journal of Universal Computer Science*, **2**(1).

Together Soft. 2002. *Together Case Tool.*

Troyer, O. De. 2001. *Information Modelling in the new millenium.* M. Rossi and K. Siau (eds.) IDEA Group Publishing. Chap. Audience-driven web design.

Troyer, O.M.F. De, & Leune, C.J. 1998. WSDM: a user centered design method for Web sites. *In: Seventh International World Wide Web Conference.* Springer-Verlag. Lecture Notes in Computer Science.

UDDI. 2001. *Universal Description, Discovery and Integration.* http://uddi.microsoft.com/.

UIML. 2000. *User Interface Modeling Language.* http://www.uiml.org.

UML 1.4. 2001 (09). *OMG Unified Modelling Language Specification.* http://www.omg.org/technology/documents/formal/uml.htm.

VanderMeer, D., Dutta, K., & Datta, A. 2000. Enabling Scalable Online Personalization on the Web. *In: ACM Conference on Electronic Commerce (EC'00).* ACM.

Visible Analyst. 2002. *Visible Analyst Case Tool.*

W3C. 2000. *WWW Consortium.* http://www.w3.org/.

WebE. 2002. *The WebEngineering.org community site.* www.webengineering.org.

WebRatio. 2002. *Web Ratio Design Tool Suite.* http://www.webratio.com.

WSDL. 2001. *Web Services Description Language.* http://www.w3.org/TR/wsdl.

Wu, H. 2001. *A Reference Architecture for Adaptive Hypermedia Systems.*

XAML. 2001. *Transaction Authority Markup Language (White Paper).* www.xaml.org.

XLANG. 2001. *Web Services for Business Process Design.* www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.html.

XMI. 1999. *XML Metadata Interchange.* http://www-4.ibm.com/software/ad/library/standards/xmi.html.

XML. 2000. *eXtensible Markup Language.* http://www.w3.org/XML/.

Y. Wadsworth. 2001. *What is Participatory Action Research?* http://www.scu.edu.au/schools/sawd/ari/ari-wadsworth.html.

Volver al Índice/Tornar a l'índex

Yoo, J., & Bieber, M. 2000 (01). Towards a Relationship Navigation Analysis. *In: Prooceedings of the 33rd Hawaii International Conference on System Sciences.*

A. DTD's correspondientes a la taxonomía de plantillas de OO-H

Este apéndice muestra los DTD's correspondientes a las distintas plantillas que permiten la obtención de una descripción textual de la interfaz objeto de modelo, tal y como vimos en el capítulo 8. Estas plantillas sirven además de base a los generadores incluidos como parte del entorno de desarrollo de OO-H (ver capítulo 9) y, por tanto, incluyen en su versión final información necesaria para este proceso de generación, que va más allá de la especificación textual obtenida a partir del DAN, del DPA y/o del DDV.

A continuación presentamos cada uno de dichos DTD's.

A.1 Plantilla de contenido de páginas

```
<!ELEMENT tForm (collection|form)*> <!ATTLIST tForm
    id ID #REQUIRED
    idGeneratorLink CDATA #IMPLIED
    mainPage (true|false) "false"
    defaulttRuleAssociated CDATA #IMPLIED
>

<!-- Con respecto a los formularios: --> <!ELEMENT collection
(object)*> <!ATTLIST collection
    id ID #REQUIRED
    idGeneratorLink CDATA #IMPLIED
    query_associated (true|false) "false"
>

<!ELEMENT object (form|collection|attrib)*> <!ATTLIST object
    id ID #REQUIRED
    class CDATA #REQUIRED
>

<!ELEMENT attrib EMPTY> <!ATTLIST attrib
    id ID #REQUIRED
    type (string|boolean|number|date|time|file|uri|other) #REQUIRED
>

<!-- Con respecto a los formularios: --> <!ELEMENT form
(action,(field|collection)*, submit, reset?)> <!ATTLIST form
```

```
    id ID #REQUIRED
    extraParams CDATA #REQUIRED
    virtualParent CDATA #IMPLIED
>

<!ELEMENT action EMPTY> <!ATTLIST action
    dest CDATA #REQUIRED
    method (get|post) "post"
    logicPath CDATA #REQUIRED
>

<!ELEMENT field (interactionType, (range |
(string|boolean|number|date|time|file|uri|other)), mask?)>
<!ATTLIST field
    id CDATA #REQUIRED
    alias CDATA #IMPLIED
    validate (onChange|onSubmit|no) "no"
    mandatory (yes|no) "no"
>

<!ELEMENT interactionType EMPTY> <!ATTLIST interactionType
    id CDATA #REQUIRED
    type (select|input) "input"
    default CDATA #IMPLIED
>

<!ELEMENT range (allowed+|defAllowed|filter)> <!ELEMENT allowed
EMPTY> <!ATTLIST allowed
    id CDATA #IMPLIED
    value CDATA #IMPLIED
>

<!ELEMENT filter EMPTY> <!ATTLIST filter
    id CDATA #IMPLIED
>

<!ELEMENT defAllowed EMPTY> <!ATTLIST defAllowed
    minAllowed CDATA #IMPLIED
    maxAllowed CDATA #IMPLIED
>

<!ELEMENT string (mask)*> <!ATTLIST string
    minsize CDATA #IMPLIED
    maxsize CDATA #IMPLIED
    value CDATA #IMPLIED
>

<!ELEMENT boolean EMPTY> <!ATTLIST boolean
    name CDATA #IMPLIED
    value (undefined|yes|no) "undefined"
>

<!ELEMENT number (real|integer|money)> <!ATTLIST number
```

```
    minvalue CDATA #IMPLIED
    maxvalue CDATA #IMPLIED
    value CDATA #IMPLIED
>

<!ELEMENT integer EMPTY>

<!ELEMENT real EMPTY> <!ATTLIST real
    decimals CDATA #IMPLIED
>
<!ELEMENT money EMPTY> <!ATTLIST money
    decimals CDATA #IMPLIED
    currency CDATA #IMPLIED
>

<!ELEMENT date EMPTY> <!ATTLIST date
    minvalue CDATA #IMPLIED
    maxvalue CDATA #IMPLIED
    precision (years|months|days) "years"
>

<!ELEMENT time EMPTY> <!ATTLIST time
    minvalue CDATA #IMPLIED
    maxvalue CDATA #IMPLIED
    precision (hours|minutes|seconds) "hours"
    zone CDATA #IMPLIED
>

<!ELEMENT file EMPTY> <!ATTLIST file
    minsize CDATA #IMPLIED
    maxsize CDATA #IMPLIED
    type (img|text|doc|zip|other) "other"
    download (yes|no) "yes"
>


<!ELEMENT uri EMPTY> <!ATTLIST uri
    minsize CDATA #IMPLIED
    maxsize CDATA #IMPLIED
    protocol (http|ftp|mailto|other|none) "http"
>

<!ELEMENT mask EMPTY> <!ATTLIST mask
    visible CDATA "yes"
    examble CDATA "implied"
>

<!ELEMENT submit EMPTY> <!ATTLIST submit
    id ID #REQUIRED
>
<!ELEMENT reset EMPTY> <!ATTLIST reset
    id ID #REQUIRED
>
```

A.2 Plantilla de enlazado de páginas

```
<!ELEMENT tLink (APDLink*)>

<!ELEMENT APDLink ((APDElement|APDArc)*)> <!ATTLIST APDLink
    id ID #REQUIRED
    role NMTOKEN #IMPLIED
    title CDATA #IMPLIED
    browsing (False|True) "False"
    navigationPages CDATA #IMPLIED
    objectsPerPage CDATA #IMPLIED
>

<!ELEMENT APDElement (SelectElement)*> <!ATTLIST APDElement
    id CDATA #REQUIRED
    type (tForm|tFunction|tStyle|tExternal) #REQUIRED
    href CDATA #REQUIRED
    parent CDATA #IMPLIED
    role NMTOKEN #IMPLIED
    title CDATA #IMPLIED
>

<!ELEMENT APDArc (lFunction)* > <!ATTLIST APDArc
    type (arc|dep|resp) <!--"arc" salto nav por un link, resp es que vienes de
    formulario-->
    from NMTOKEN #REQUIRED
    to NMTOKEN #REQUIRED
    show (new|replace|embed|other|none) "replace"
    actuate (onLoad|onRequest|other|none) "onRequest"
    orgForm CDATA #IMPLIED <!--formulario del que procede.-->
    filter CDATA #IMPLIED
    service CDATA #IMPLIED
>

<!ELEMENT SelectElement EMPTY> <!--params--> <!ATTLIST
SelectElement
    name CDATA #REQUIRED
    type (inside|outside) #FIXED "inside"
    domain CDATA #IMPLIED
>

<!ELEMENT apdDep (lSource,lDest) > <!ATTLIST apdDep
    type (arc|dep) #FIXED "dep"
    show (merge|reference) "merge"
>

<!ELEMENT lSource EMPTY> <!ATTLIST lSource
    role NMTOKEN #IMPLIED
>

<!ELEMENT lDest EMPTY> <!ATTLIST lDest
    role NMTOKEN #IMPLIED
```

```
>

<!ELEMENT lFunction EMPTY> <!ATTLIST lFunction
    name ID #REQUIRED
        event (onLoad|onBlur|onActivate) #REQUIRED
>
```

A.3 Plantilla de composición de páginas

```
<!-- Estructura general del CLD: zonas--> <!-- Una sola pagina
tLayout para cada APD -->

<!ELEMENT tLayout (tLayoutGroup)*>

<!ELEMENT tLayoutGroup (tLayoutElement)*> <!ATTLIST tLayoutGroup
    id CDATA #REQUIRED
    title CDATA #IMPLIED
>

<!ELEMENT tLayoutElement (tReference)*> <!ATTLIST tLayoutElement
    id ID #REQUIRED
    left CDATA #REQUIRED
    top CDATA #REQUIRED
    height CDATA #REQUIRED
    width CDATA #REQUIRED
    color CDATA #IMPLIED
    transparent (true|false) "true"
    sizeable (true|false) "false"
    scrollable (true|false) "false"
    popup (true|false) "false"
>

<!-- Referencias a paginas tStruct o tForm asociadas al layout -->

<!ELEMENT tReference EMPTY> <!ATTLIST tReference
    id ID #REQUIRED
>
```

A.4 Plantilla de coordenadas de localización de elementos

```
<!ELEMENT tLocation (tPageReferenced)*>
```

```
<!ELEMENT tPageReferenced (tLocationElement*)> <!ATTLIST
tPageReferenced
    id ID #REQUIRED
>

<!ELEMENT tLocationElement EMPTY> <!ATTLIST tLocationElement
    id CDATA #REQUIRED
    left CDATA #REQUIRED
    top CDATA #REQUIRED
    height CDATA #REQUIRED
    width CDATA #REQUIRED
>
```

A.5 Plantilla de estilos de elementos de página

```
<!ELEMENT tStyle (import*, rule*)> <!ATTLIST tStyle
    type CDATA "web"
    device CDATA "screen"
>

<!ELEMENT import EMPTY> <!ATTLIST import
    url CDATA #REQUIRED
>

<!ELEMENT rule (rendition)> <!ATTLIST rule
    id CDATA #REQUIRED
    description CDATA #IMPLIED
>

<!ELEMENT rendition
(display?,list?,font?,margin?,padding?,text?,background?,color?,border?,zindex?,float?)>

<!ELEMENT display EMPTY> <!ATTLIST display
    format (external|table|table-row|table-cell|table-row-group|table-header-group|table-foote
    transparent (true|false) #IMPLIED
>

<!ELEMENT list EMPTY> <!ATTLIST list
    type (ulist|olist) #IMPLIED
    expandDirection (left|right|up|down) #IMPLIED
>

<!ELEMENT font EMPTY> <!ATTLIST font
    family CDATA #IMPLIED
    unit NMTOKEN #IMPLIED
    size CDATA #IMPLIED
    weight CDATA #IMPLIED
    bold (true|false) #IMPLIED
```

```
    italic (true|false) #IMPLIED
    underline (true|false) #IMPLIED
    color CDATA #IMPLIED
>

<!ELEMENT margin EMPTY> <!ATTLIST margin
    unit (px|cm) #IMPLIED
    bottom CDATA #IMPLIED
    top CDATA #IMPLIED
    left CDATA #IMPLIED
    right CDATA #IMPLIED
>

<!ELEMENT  padding EMPTY> <!ATTLIST padding
    unit (px|cm) #IMPLIED
    bottom CDATA #IMPLIED
    top CDATA #IMPLIED
    left CDATA #IMPLIED
    right CDATA #IMPLIED
>

<!ELEMENT text EMPTY> <!ATTLIST text
    unit NMTOKEN #IMPLIED
    align (left|right|center) #IMPLIED
    height CDATA #IMPLIED
    indent CDATA #IMPLIED
>

<!ELEMENT  background EMPTY> <!ATTLIST background
    color CDATA #IMPLIED
    complementaryColor CDATA #IMPLIED
    src CDATA #IMPLIED
    pattern (mosaic|justified|single) #IMPLIED
>

<!ELEMENT color EMPTY> <!ATTLIST color
    name CDATA #IMPLIED
>

<!ELEMENT border EMPTY> <!ATTLIST border
    visible (true|false) #IMPLIED
    unit (px|cm) #IMPLIED
    width CDATA #IMPLIED
    type (normal|inset|outset) #IMPLIED
    color CDATA #IMPLIED
>

<!ELEMENT zindex EMPTY> <!ATTLIST zindex
    stackLevel CDATA #IMPLIED
>

<!ELEMENT  float EMPTY> <!ATTLIST float
    align (right|left) #IMPLIED
```

```
>
```

A.6 Plantilla de referencias a elementos externos

```
<!-- Referencias a elementos que poseen un estilo determinado -->

<!ELEMENT tReference (tPageReferenced)*>

<!ELEMENT tPageReferenced
(tReferenceElement*|tReferenceAssociation*)> <!ATTLIST
tPageReferenced
    id ID #REQUIRED
    defaulttStyleRule CDATA #IMPLIED
>


<!ELEMENT tReferenceElement (tReferenceTextIOL)*> <!ATTLIST
tReferenceElement
   id CDATA #REQUIRED
   defaulttStyleRule CDATA "default"
   text CDATA #IMPLIED
   src  CDATA #IMPLIED
   tipText CDATA #IMPLIED
   type (collection|form|img|text|textbox|datebox|textarea|file|button|combobox|radiobutton|ch
   typeAtributes CDATA #IMPLIED
   external (true|false) #IMPLIED
>

<!ELEMENT tReferenceTextIOL EMPTY> <!ATTLIST tReferenceTextIOL
   text CDATA #REQUIRED
   languaje (spanish|english|french|german|italian|valenciano) #REQUIRED
>

<!ELEMENT tReferenceAssociation EMPTY> <!ATTLIST
tReferenceAssociation
   idElement CDATA #REQUIRED
   idAssociation CDATA #REQUIRED
>
```

B. Un ejemplo de aplicación final generada: Aula Virtual

A continuación, y con el fin de proporcionar un ejemplo de aplicación obtenida como resultado de aplicar el método OO-H, presentamos un Aula Virtual. Esta aplicación está disponible en la dirección *http* : *//visualwade.dlsi.ua.es/colegio2/*.

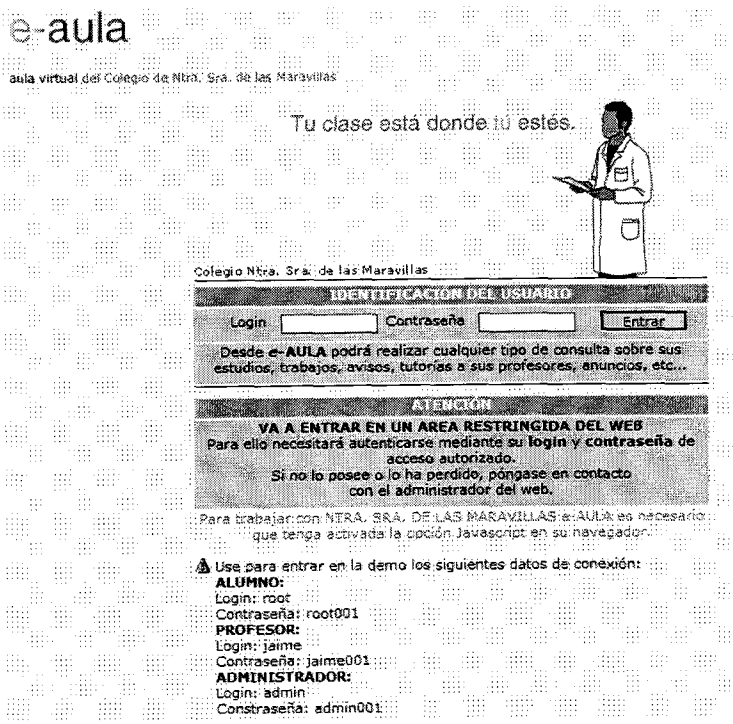

Figura B.1. *Autentificación*

La primera pantalla de la aplicación incluye un formulario de identificación. En esta pantalla (ver Fig. B.1) se puede ver que se han definido tres tipos de actores, cada uno con una vista distinta asociada:

- Vista alumno
- Vista profesor
- Vista administrador

La aplicación dispone de un control de acceso. Si, durante la identificación del usuario, éste no es reconocido por el sistema, la aplicación muestra un mensaje de error como el presentado en la Fig. B.2.

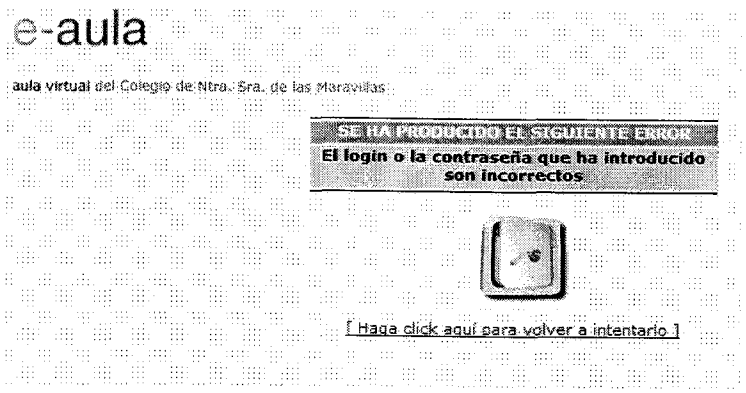

Figura B.2. *Error de autentificación*

Una vez el usuario se ha autentificado, en función de su perfil se accederá a una de las vistas definidas. A partir de ahora, supondremos que nos hemos identificado como un alumno.

Los alumnos validados acceden a una página donde se muestran los avisos de última hora relevantes para ese usuario. También se puede solicitar la visualización del histórico de avisos recibidos, mediante la activación del enlace indicado, como se puede ver en la Fig. B.3, que nos conduce a una página como la que puede observarse en la Fig. B.4.

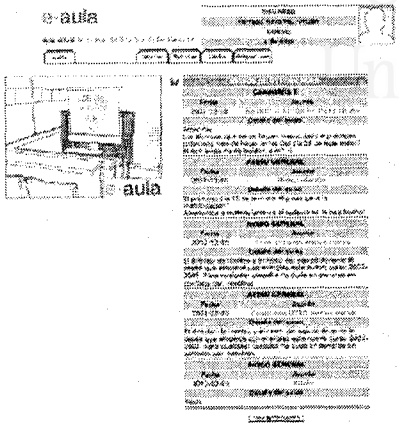

Figura B.3. *Home - Histórico de avisos*

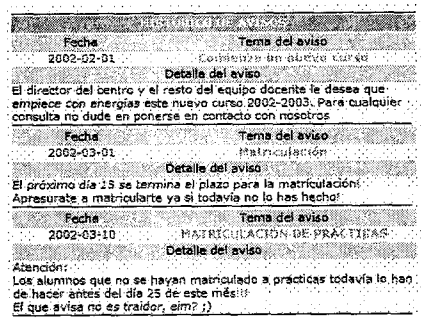

Figura B.4. *Home - Avisos anteriores*

Desde el menú podemos acceder al apartado de *Tutorías* (ver Fig. B.5).

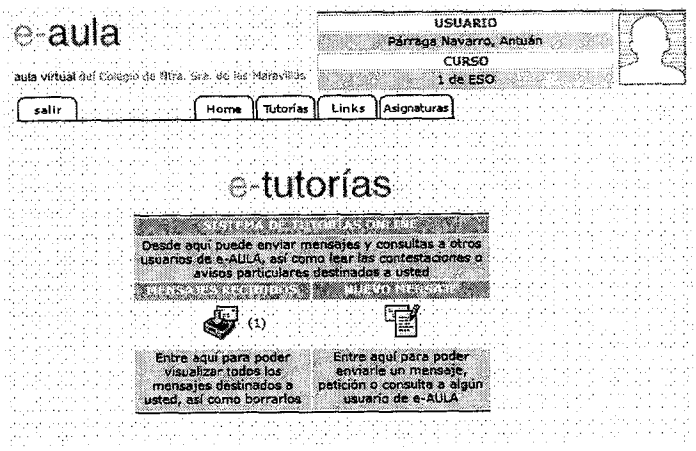

Figura B.5. *Tutorías*

Desde esta pantalla es posible tanto visualizar los mensajes recibidos (ver Fig. B.6) como enviar un nuevo mensaje (ver Fig. B.7).

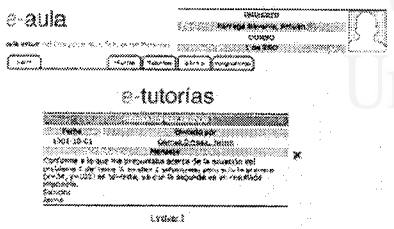

Figura B.6. *Mensajes recibidos*

Figura B.7. *Seleccionar receptor del mensaje*

Si se selecciona la opción *Ver mensajes recibidos* se mostrará una lista con todos los mensajes que se han recibido. Si se opta por escribir un nuevo mensaje, se mostrará una lista con los nombres de los profesores. Al seleccionar la opción *enviar mensaje* a uno de los profesores, aparecerá una nueva pantalla donde podremos redactar y enviar el mensaje deseado (ver Fig. B.8).

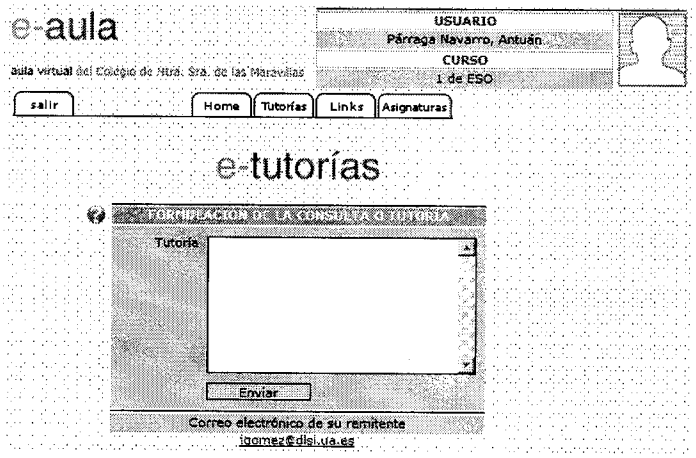

Figura B.8. *Enviar mensaje*

La siguiente opción del menú, *Links*, nos muestra una lista de enlaces interesantes que podemos visitar (ver Fig. B.9).

Por último, podemos ver información interesante de las asignaturas ofertadas, que primeramente se muestran en una lista (ver Fig.

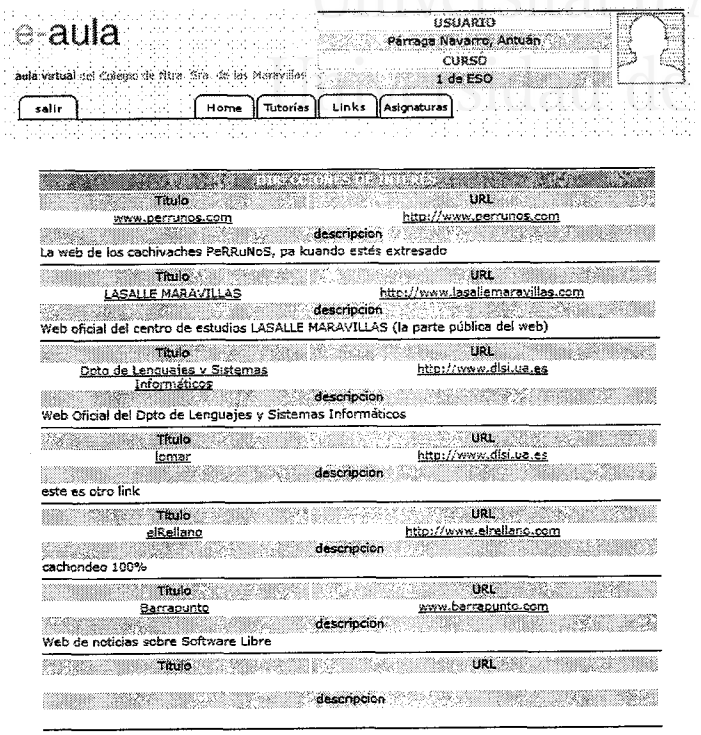

Figura B.9. *Enlaces de interés*

B.10). Al elegir una de ellas, accedemos a la información concreta de esa asignatura, tal y como podemos ver en la Fig. B.11.

Una vista general de la estructura de la aplicación que interrelaciona todas estas páginas puede ser vista en la Fig. B.12.

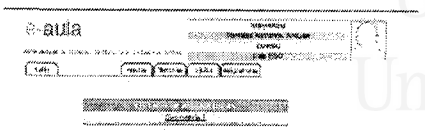

Figura B.10. *Listado de asignaturas*

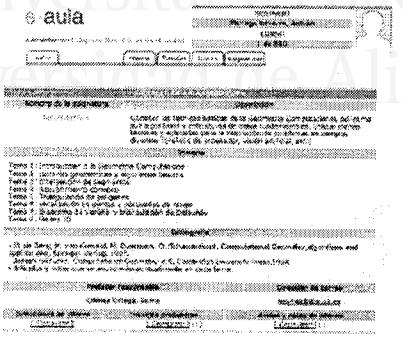

Figura B.11. *Datos de la asignatura*

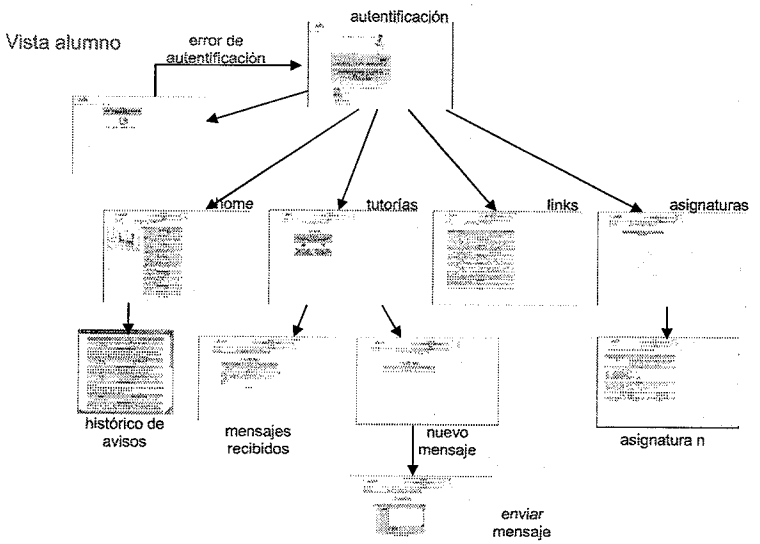

Figura B.12. *Esquema general de las páginas generadas para el perfil* Alumno

Volver al Índice/Tornar a l'índex

C. La Autora

Cristina Cachero Castro nació en 1973 en Alicante, España. Obtuvo el título de Ingeniería en Informática por la Universidad de Alicante en 1996, tras haber trabajado durante un año como administradora de bases de datos en Liège (Bélgica). De 1996 a 1999 trabajó en el Laboratorio Multimedia, adscrito a la Universidad de Alicante, como analista de sistemas y programadora en Internet. En 1999, con motivo de la obtención de una beca F.P.I. de la Conselleria de Cultura, Educació y Ciència, pasó a formar parte del Departamento de Lenguajes y Sistemas Informáticos, donde actualmente ejerce labores docentes y de investigación. En el año 2000 realizó una estancia de 3 meses en el Politecnico de Milano (Italia). Sus publicaciones pueden ser consultadas en *http : //www.dlsi.ua.es/ ccachero/pPublicaciones.htm.*