Modelando aspectos de navegación y presentación en aplicaciones hipermediales

Cristina Cachero[1⋆], Jaime Gómez[1], and Oscar Pastor[2]

[1] Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante. SPAIN
{ccachero,jgomez}@dlsi.ua.es
[2] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia. SPAIN
opastor@dsic.upv.es

Abstract Este artículo presenta OO-$\mathcal{H}$Method, una propuesta metodológica que, utilizando una aproximación OO, captura la semántica necesaria para el modelado eficiente de Interfaces de Usuario en web, y extiende por tanto los métodos existentes de modelado conceptual de aplicaciones. OO-$\mathcal{H}$Method recoge las dimensiones de navegación y presentación características de este tipo de aplicaciones mediante dos nuevos diagramas: el Diagrama de Acceso Navegacional (DAN) y el Diagrama de Presentación Abstracta (DPA). El DAN toma como base un diagrama de clases simplificado que contiene la organización de la información manejada por cada tipo de usuario, y lo enriquece con los modos de acceso y navegación deseados. A partir de él, y aplicando una serie de reglas de transformación, se genera un DPA por defecto. El DPA se define como una estructura de plantillas especificadas en XML. Estas plantillas pueden ser editadas y refinadas por el diseñador para conseguir los rasgos de interfaz deseados. Para enriquecer y refinar los diagramas OO-$\mathcal{H}$Method integra un Catálogo de Patrones de Interfaz, que además captura reglas de diseño que ayudan a incrementar la calidad y facilidad de uso de la misma. Como resultado, una interfaz de aplicación web integrable con módulos de lógica preexistentes puede ser generada de forma automática a partir de esta especificación.

1 Introducción

La rápida evolución de Internet en general y del WWW en particular ha propiciado en los últimos años la investigación intensiva en el campo del modelado conceptual de aplicaciones web. En este contexto se han propuesto distintos métodos, lenguajes y herramientas de modelado hipermedial. Algunos de los más relevantes estudiados hasta el momento son HDM [9], HDM-lite [7], WebML[4], OOHDM [22], RMM [12], ADM [1, 14] o Strudel [6]. Sin embargo estos métodos se centran en su mayor parte en aspectos de navegación y presentación, obviando los relacionados con la funcionalidad y las características de interacción específicas de este tipo de aplicaciones.

En este contexto, nuestros esfuerzos no se han centrado en la propuesta de 'un nuevo método de modelado de aplicaciones en web', sino en definir la semántica y notación que permita el desarrollo de interfaces basadas en web y su conexión con lógica de aplicación previamente especificada. Para ello nuestra propuesta, conocida como OO-$\mathcal{H}$Method[11, 3], parte del diagrama de clases de modelos OO especialmente diseñados para tal efecto, como pueden ser OO-Method [19, 20] o UML [17]. OO-$\mathcal{H}$Method extiende los modelos OO convencionales con dos nuevos diagramas: (1) el Diagrama de Acceso Navegacional (DAN) y (2) el Diagrama de Presentación Abstracta (DPA). Tanto el DAN como el DPA capturan parte de la información relevante para el diseño de la interfaz por medio de un conjunto de patrones definidos en un Catálogo de Patrones de Interfaz[2] que es parte integrante de OO-$\mathcal{H}$Method.

El resto del artículo está estructurado como sigue: la sección 2 proporciona una breve descripción del Catálogo de Patrones de OO-$\mathcal{H}$Method, que actúa como repositorio de soluciones aplicables tanto a nivel de DAN como a nivel de DPA. La sección 3 proporciona, mediante un ejemplo, una visión general de los aspectos semánticos y estructurales del DAN, que es el encargado de captura los requerimientos navegacionales del usuario. El DAN sirve también de base para la generación del DPA por defecto, que captura los rasgos básicos de la interfaz. La sección 4 introduce los conceptos y los constructores de

⋆ Este artículo ha sido escrito con el patrocinio de la Conselleria de Cultura, Educació i Ciència de la Comunitat Valenciana

plantilla utilizados en la construcción del DPA. También define el proceso de transformación entre ambos diagramas, así como el modo de aplicar patrones para refinar la interfaz web resultante. Para ilustrar el proceso, la interfaz generada a partir del ejemplo se muestra en la sección 5. Por último, la sección 6 realiza una comparación con el trabajo relacionado, y la sección 7 presenta las conclusiones y los trabajos futuros.

2 El Catálogo de Patrones de Interfaz de OO-$\mathcal{H}$Method

El Catálogo de OO-$\mathcal{H}$Method constituye un Lenguaje de Patrones de Interfaz, es decir, una colección parcialmente ordenada de guías, en un formato estándar, para dar solución a problemas recurrentes en el ámbito del diseño de Interfaces Hipermediales [21, 10, 18]. En nuestro caso la recopilación y catalogación de patrones hipertextuales ayudan explícitamente a capturar el modelo de interacción abstracta entre el usuario y la aplicación.

El catálogo está estructurado de la siguiente forma:

1. Patrones de Información: son patrones que capturan formas de transmitir al usuario información acerca de su contexto de interacción. Como ejemplo se puede citar el 'Patrón de Localización' [21], que ofrece soluciones al síndrome clásico de 'perdido en el hiperespacio'.
2. Patrones de Interacción: abordan problemas relacionados con la introducción de información por parte del usuario en tiempo de ejecución. Un ejemplo es el 'Patrón de filtro de Población', que proporciona una serie de mecanismos alternativos para la restricción del conjunto de objetos con los que el usuario desea trabajar.
3. Patrones de Navegación: definen el modo en que el usuario atraviesa la información. Son patrones particularmente relevantes en el ámbito de la metáfora del hipertexto.

Otra de las características de nuestro catálogo es que es, como el resto del modelo, 'Orientado a Usuario', es decir, recoge soluciones a los problemas de uso de la interfaz. Su existencia permite al diseñador elegir, de entre un conjunto de implementaciones alternativas, la más apropiada dependiendo del dominio y del tipo de usuario considerado. Además nuestro catálogo no es un conjunto cerrado, sino que va evolucionando y ampliándose según evoluciona nuestro método.

En OO-$\mathcal{H}$Method, cada patrón tiene un ámbito de aplicación asociado. Existen patrones relacionados con la selección de información y con el comportamiento navegacional del usuario. Estos patrones son aplicables a nivel de DAN. Sin embargo, existen otros que pueden ser considerados 'patrones de refinamiento', es decir, patrones que proporcionan funcionalidad añadida a la interfaz. Este tipo de patrones se aplica a nivel de DPA.

A continuación vamos a explicar los principales aspectos semánticos y estructurales del DAN, cuya definición es el primer paso en el proceso de modelado de la interfaz.

3 El Diagrama de Acceso Navegacional de OO-$\mathcal{H}$Method

Para dar una visión más general de nuestra aproximación, se va a introducir un ejemplo que será retomado a lo largo de todo el artículo según se vayan introduciendo nuevos conceptos: un Gestor de Foros de Discusión. Como explicación básica (por razones de brevedad) asumiremos que la aplicación gestiona varios temas de discusión (Java Server Pages, Servlets, etc), y que los mensajes se refieren siempre a un solo tema. Además los mensajes están estructurados jerárquicamente de manera que un mensaje puede ser el inicio de una nueva línea de discusión dentro de su tema o ser una respuesta a otro mensaje previamente existente. El tipo de usuario cuyo comportamiento pretendemos modelar podrá leer mensajes, crear una nueva línea de discusión o contestar a otro mensaje.

OO-$\mathcal{H}$Method asocia un DAN diferente para cada tipo de usuario (agente). Este diagrama se basa en los siguientes constructores:

1. Clases Navegacionales (CN): son clases de dominio cuyos atributos y métodos han sido filtrados y enriquecidos con información adicional, necesaria para la correcta especificación de determinados rasgos de las interfaces hipertextuales. Un ejemplo de enriquecimiento de clase es la diferenciación entre

tres tipos de atributos atendiendo a su visibilidad: V-Atributos (atributos que siempre están visibles), R-Atributos (atributos de los que sólo aparece una referencia, y que requieren una demanda explícita del usuario para ser visualizados) y H-Atributos (atributos raramente relevantes para ese usuario y esa vista, y que por tanto, y para mantener una interfaz limpia, se ocultan totalmente y sólo se muestran en vistas exhaustivas del sistema).

En nuestro ejemplo (ver Fig. 1) se han identificado dos clases de dominio: la clase 'Chat', que determina los temas de discusión activos en el Gestor de Foros, y la clase 'Messages', que contiene los mensajes que guarda el sistema. Vemos que la clase 'Chat' tiene un solo atributo de tipo V-atributo (siempre visible). Dicho atributo es el 'nameChat' y especifica el nombre identificador de cada tema de discusión. La clase 'Messages' tiene, además de su atributo identificador 'titleMsg' (descripción corta del tema del mensaje), un campo llamado 'textMsg' de tipo R-Atributo, lo que implica que el usuario tendrá que demandar específicamente su visualización para que le aparezca en pantalla. En nuestro caso esta demanda consistirá en activar el enlace definido sobre 'titleMsg' (ver Fig. 5).

2. Destinos Navegacionales (DN): agrupan los elementos del modelo que colaboran en la cobertura de los distintos requerimientos navegacionales del usuario.

   Tomando nuevamente nuestro ejemplo como referencia (ver Fig. 1), vemos que tenemos, debido a su simplicidad, un solo requerimiento navegacional: participar en el foro de discursión. Esto se ha modelado mediante el Destino Navegacional 'Participate in Chat', que agrupa tanto la información como los servicios requeridos por el usuario.

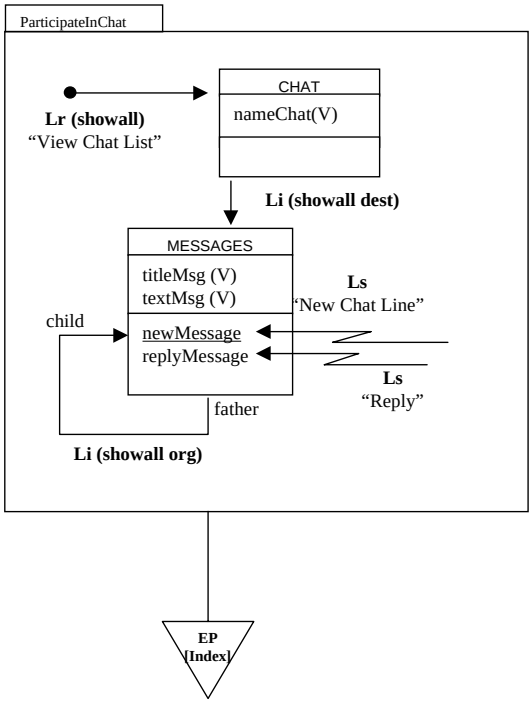


Figure1. *DAN del Gestor de Foros de Discusión*

3. Enlaces de Navegación (EN): definen los caminos de navegación que puede seguir un usuario a través de la información. Tienen un conjunto de Patrones de Navegación (definidos en el Catálogo de Patrones) y un conjunto de Filtros de Navegación asociados, que cualifican el comportamiento navegacional y proporcionan la información adicional necesaria para definir el modelo de navegación del usuario. Podemos distinguir entre cuatro tipos de enlaces: I-Links (Enlaces Internos), que definen caminos de navegación entre objetos dentro de un DN determinado, T-Links (Enlaces de Travesía), que se definen entre clases de navegación que pertenecen a DN distintos, R-Links (Enlaces de Requerimiento), que definen el punto de entrada a cada DN y S-Links (Enlaces de Servicio), que determinan

los servicios que se encuentran disponibles para el tipo de usuario asociado a ese DAN.

En la Fig. 1 se observan tres de los cuatro tipos de enlaces posibles: Li (abreviatura de I-Links), Lr (R-Links) y Ls (S-Links). La relación estructural de composición existente entre las clases 'Chat' y 'Messages', aunque queda oculta en el DAN, sí hace posible sin embargo la aparición de un enlace de navegación entre dichas clases. El enlace es interno porque no salimos del DN en el que nos encontramos. Además, el hecho de que este enlace no tenga un nombre asociado provoca su materialización mediante el atributo 'nameChat' de la clase 'Chat' (ver Fig. 4). Este enlace tiene asociado el patrón de navegación 'showall dest', que provoca la visualización de todas las instancias de la clase en la misma vista. El hecho de que sea en destino (dest) indica que para ver los mensajes se debe dar un salto en la navegación. En nuestro caso, este salto se materializará en última instancia como un enlace a una nueva página. El Lr (enlace de requerimiento) 'View Chat List' muestra el punto de entrada al Destino Navegacional, es decir, el punto desde el cual se va a iniciar la navegación. Un ejemplo de Ls es el asociado con el servicio 'replyMessage' de la clase 'Messages', que es el que posibilita que el usuario pueda contestar a mensajes previamente enviados al foro. Cada vez que se muestre un objeto de tipo 'Messages', aparecerá la opción de contestar a ese objeto. El servicio replyMessage tiene, aunque no aparezca reflejado en el diagrama, una serie de parámetros asociados y un valor booleano de retorno que indica si la inserción del mensaje ha sido realizada correctamente. Los parámetros, como no son proporcionados por el modelo, deberán ser introducidos en tiempo de ejecución por el usuario mediante un formulario que se genera automáticamente. El valor de retorno se traslada al usuario mediante una pantalla de confirmación/error de ejecución de servicio. Todas estas características se pueden observar en la Fig. 5.

4. Colecciones (C): son estructuras jerárquicas que se definen sobre clases o destinos navegacionales, y especifican formas de acceso a la información . Al igual que los EN, tienen asociados un conjunto de Patrones de Navegación y de Filtros de Navegación que definen tanto el comportamiento de travesía de cada uno de los enlaces agrupados bajo esa colección como el conjunto de objetos sobre los cuales se aplicará dicha colección. En OO-$\mathcal{H}$Method podemos distinguir varios tipos de colecciones, entre las que destacamos (1) C-Colecciones (Colecciones de Clasificación), que determinan una estructura de acceso, jerárquica o no, a la información o servicios disponibles y (2) S-Colecciones (Colecciones de Selección), que agrupan objetos que cumplen las condiciones introducidas por el usuario en tiempo de ejecución mediante un formulario generado automáticamente a partir de los parámetros especificados en el DAN

   En nuestro ejemplo (ver Fig. 1) tenemos una sola colección, dibujada como un triángulo invertido y asociada a nuestro DN. Esta C-Colección es especial, ya que determina el punto de entrada a la aplicación (de ahí que esté identificada como EP, del inglés 'Entry Point'). Como no tiene nombre asociado, los enlaces que agrupa (en este caso sólo uno) utilizarán como identificador el nombre de los correspondientes Lr, definido en cada DN.

Para más información sobre este diagrama, los lectores interesados pueden acudir a [11].

El DAN captura los caminos de navegación y los servicios que el usuario puede activar cuando está trabajando con la interfaz, y por tanto debe ser definido por separado para cada tipo de usuario. A partir de este diagrama se puede generar, mediante una serie de reglas de transformación, una interfaz web por defecto. Este rasgo del DAN permite al diseñador acortar el tiempo necesario para desarrollar prototipos de aplicaciones. Aún así, normalmente las implementaciones finales de interfaces de usuario requieren una mayor sofisticación tanto a nivel de apariencia como a nivel de rasgos de facilidad de uso. Para ello, OO-$\mathcal{H}$Method define un diagrama de diseño, el DPA, que será definido en la siguiente sección.

4 El Diagrama de Presentación Abstracta de OO-$\mathcal{H}$Method

El DPA permite el refinamiento de la estructura de interfaz y de los rasgos de visualización capturados en las etapas previas del modelo. Concretamente, 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 un DPA por defecto. Para ayudar al diseñador a refinar esta estructura manteniendo al mismo tiempo su calidad, el Catálogo de Patrones contiene una serie de construcciones de probada

eficacia para la solución de los distintos problemas identificados hasta el momento en el ámbito de las aplicaciones hipermediales.

Hemos adoptado una aproximación basada en plantillas [1, 6, 7, 14] para la especificación tanto de la apariencia visual como de la estructura de página de la interfaz hipermedial. El uso de plantillas favorece la reutilización de las experiencias de diseño y facilita la consistencia tanto dentro de los distintos módulos que componen la aplicación como entre distintas aplicaciones.

4.1 Tipos de Plantillas

OO-$\mathcal{H}$Method define cinco tipos de plantillas, expresadas como documentos XML (eXtensible Markup Language) [5, 13]. XML permite al diseñador mantener información acerca del significado semántico de los distintos datos capturados en OO-$\mathcal{H}$Method. Para ello especifica un marco estandarizado dentro del cual definir las etiquetas necesarias para capturar dicha información. Las páginas resultantes pueden ser directamente visualizadas o traducidas a cualquier otro lenguaje para su publicación. Con esta aproximación, la adición de nuevos tipos de plantillas es muy sencilla.

Las etiquetas y la estructura del documento se han definido mediante un conjunto de DTD's (Document Type Definition), uno para cada tipo de plantilla. El DTD especifica el conjunto de reglas que debe cumplir un documento XML para ser considerado válido. Como las páginas abstractas son documentos XML, el DTD especificará las etiquetas que puede contener ese tipo de página, las anidaciones válidas de etiquetas, sus atributos y los valores permitidos. También contiene otro tipo de información, como son instrucciones de proceso, comentarios, declaración de tipo de documento etc.

Los tipos de plantilla definidos hasta el momento en OO-$\mathcal{H}$Method son:

1. tStruct: estructuran la información que debe aparecer en la página materializada.
2. tStyle: definen los rasgos de visualización como son el emplazamiento físico de los elementos, la tipografía o la paleta de color.
3. tForm: definen las estructuras de información que debe introducir el usuario para interactuar con el sistema.
4. tFunction: capturan funcionalidad de cliente mediante funciones definidas basándose en el modelo DOM (Document Object Model)[5].
5. tWindow: su uso implica la existencia de dos o más vistas de la información activas en un mismo instante de tiempo.

Como ejemplo, mostramos a continuación el DTD para las páginas abstractas de tipo tStruct.

tStruct El DTD that define la estructura válidad de una página abstracta de tipo TStruct es:

```
<!ELEMENT collection (object+|link*)>
<!ATTLIST collection
    format (ulist|olist) "ulist"
    style CDATA #IMPLIED
>
<!ELEMENT link (call)*>
<!ATTLIST link
    name ID #REQUIRED
    type (string|button|image|menuitem|automatic) "string"
    show (here|new) "new"
    pointsTo (tForm|tWindow|tFunction|tStyle|tStruct|command) "tStruct"
    dest CDATA #REQUIRED
>
<!ELEMENT object (attrib+|call*)>
<!ATTLIST object
    type CDATA #REQUIRED
    style CDATA #IMPLIED
>
<!ELEMENT attrib (call)*>
<!ATTLIST attrib
```

```
    name ID #REQUIRED
    type (string|date|text|integer|anchor|...) #REQUIRED
    ordered (yes|no) "no"
>
<!ELEMENT call EMPTY>
<!ATTLIST call
    event (onChange|onClick|onLoad|...) #REQUIRED
    function CDATA #REQUIRED
>
<!ELEMENT label EMPTY>
<!ATTLIST label
    text CDATA
>
```

Como características principales que pueden ser extraídas de esta definición de DTD podemos citar:

1. Un tStruct está formado por un conjunto de colecciones, que deben contener al menos un objeto, y cualquier número de enlaces. Las colecciones tienen un formato por defecto asociado (lista no numerada) y podrían tener también un estilo definido por el usuario y capturado en el modelo mediante una página abstracta de tipo tStyle.
2. Un enlace puede tener cero o más llamadas a funciones que capturan lógica de cliente. Un enlace se define mediante su nombre, un tipo y un conjunto de atributos relacionados con su comportamiento, como son dónde va a a visualizar la página destino, cuál es esa página o tipo de interacción con el usuario. Sólo los enlaces con el atributo 'mostrar' (show) instanciado a 'new' son mostrados en el DPA. Un tipo de enlace 'automático' implica que su activación no dependerá de ninguna acción por parte del usuario.
3. Un objeto está formado por uno o más atributos y cero o más llamadas a función. También tiene un tipo (la clase a la cual pertenece) y un estilo asociado de forma optativa.
4. Un atributo puede tener cualquier número de llamadas asociadas. También tiene siempre asociado un nombre y un tipo. Los atributos caracterizados como 'atributos de orden' especifican el orden de los objetos dentro de una colección. Este orden, aunque puede ser conceptualmente irrelevante, sí tiene importancia en la fase de presentación, donde siempre existe y contribuye a guiar al usuario. El orden, si no se especifica nada en el modelo, se establece como 'ordenación ascendente según el conjunto de atributos identificadores del objeto'.
5. Una llamada a función, que puede estar asociada con cualquier elemento de la página abstracta, define tanto las condiciones de disparo ('onClick', 'onMouseOver' etc) como la reacción de la interfaz (función de cliente que debe ser invocada ante ese evento).
6. Por último, una página de tipo tStruct puede tener cualquier número de etiquetas que capturan texto estático que enriquece la información visualizada en la interfaz.

La estructura de plantillas por defecto puede ser obtenida a partir de la información capturada en el DAN con la ayuda de un conjunto de políticas de generación por defecto. Estas políticas de generación serán explicadas en la siguiente sección.

4.2 Diagrama de Presentación Abstracta Básico

OO-$\mathcal{H}$Method define un conjunto de reglas para la transformación de los distintos elementos contenidos en el DAN en elementos del DPA. Algunas de estas reglas son:

1. I-Links, T-Links and R-Links: se transforman en elementos de tipo link en la plantilla de tipo tStruct asociada.
2. C-Colecciones, S-Colecciones: definen árboles cuyas hojas son enlaces que apuntan a otras plantillas. Se transforman en páginas de tipo tStruct que capturan esos enlaces. Una S-Colección genera además una página de tipo tForm para que el usuario pueda introducir los valores dinámicos que determinarán los objetos incluidos en la colección.
3. S-Links: al igual que las S-Colecciones, pueden generar una página de tipo tForm si los parámetros involucrados en el servicio considerado deben ser introducidos por el usuario. En cualquier caso, la plantilla resultante debe contener algún elemento de tipo 'link' que apunte a la API de integración de la interfaz con los servicios ofrecidos por la capa lógica. Si el comando necesita devolver algún tipo de información (o incluso nuevas colecciones), se generará además otra página de tipo tStruct con dicha información.

4. R-Atributos: provocan la aparición en el diagrama de una nueva página abstracta de tipo tStruct, y la aparición de un nuevo elemento de tipo enlace que apunta a esta nueva página desde la original. La nueva plantilla contendrá todos los R-Atributos que contiene el objeto considerado junto con una referencia (conjunto de atributos semánticamente significativos) que identifique el objeto original.
5. Patrones de Navegación: todos los patrones de navegación (índices, visitas guiadas, visitas guiadas indexadas y vistas de tipo 'showall') se capturan en el DPA mediante un conjunto de elementos de tipo enlace definidos en una página abstracta de tipo tStruct.
6. Los tipos de datos definidos para cada atributo llevan asociada una validación por defecto, que se generan por medio de elementos de tipo 'call' ante un evento 'onChange'. Este evento invocará a la función de validación correspondiente, dependiendo del tipo de atributo. Las funciones requeridas se recogen en una o varias páginas de tipo tFunction.

El DPA por defecto proporciona una interfaz operativa, que puede servir como prototipo sobre el cual testar los requerimientos de usuario. Pero en las aplicaciones hipermediales esto a menudo no es suficiente. Para conseguir dotar a la aplicación de una apariencia más sofisticada, el diseñador puede realizar una serie de refinamientos sobre este diagrama que formarán parte de la interfaz finalmente generada. Siguiendo con nuestro ejemplo, las páginas abstractas Home Page, Chat List, Message View and Reply Message y sus correspondientes enlaces (ver Fig. 2) han sido derivadas de forma automática a partir del DAN. También se ha generado una página Style genérica, que se aplica por defecto a toda la estructura de plantillas. En la siguiente sección presentaremos ejemplos de posibles modos de refinamiento, y cómo dichos refinamientos hacen que la estructura de plantillas varíe para adecuarse a la presentación final.

4.3 Refinamientos del DPA

Quizás uno de los refinamientos más importantes en términos de estructura de DPA es el causado por la introducción de plantillas de tipo tWindow. Debido a la discusión (todavía abierta) sobre la conveniencia o no de dotar al usuario con vistas simultáneas de la información, hemos decidido proporcionar este rasgo como un refinamiento aplicable a discreción del diseñador. Otro grupo importante de refinamientos nos viene dado por la aplicación de una serie de patrones, recogidos en el Catálogo de Patrones de OO-$\mathcal{H}$Method y que influyen directamente en la estructura del DPA. Estos patrones proporcionan al diseñador soluciones que recogen técnicas y rasgos ampliamente validados por la comunidad hipermedial, y que por tanto aumentan la calidad de la interfaz al mismo tiempo que proporcionan una información valiosa a diseñadores noveles.

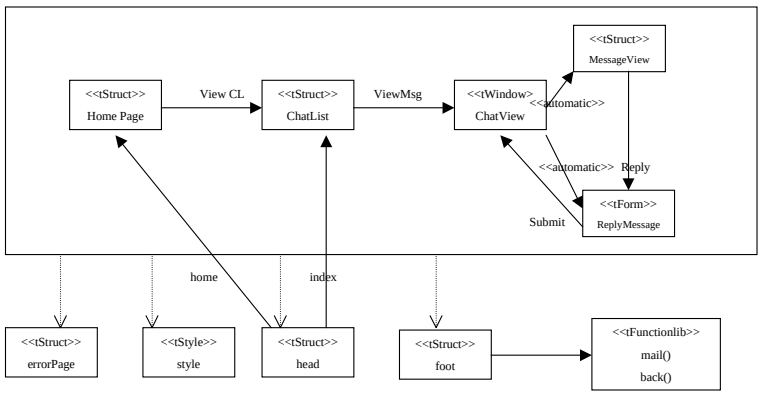


Figure2. *DPA simplificado del Gestor de Foros*

La decisión del diseñador acerca de si se debe aplicar o no cualquiera de estos patrones (y la elección entre las implementaciones disponibles) influirá en la materialización final del DPA, tanto en términos de estructura como de comportamiento. Como ejemplo, en la Fig. 2 la aplicación del 'Patrón de Localización' en su implementación más sencilla (la de inclusión de cabecera y pie, como será explicado en el siguiente párrafo) ha provocado la aparición de dos nuevas páginas abstractas ('head' y 'foot'), de tipo tStruct, así como una serie de enlaces de dependencia para indicar el conocimiento que el resto de plantillas tienen de

la existencia de estas nuevas páginas. Una definición simplificada de este patrón, extraído del Catálogo de OO-$\mathcal{H}$Method, es la siguiente:

1. Nombre: Patrón de Localización.
2. También conocido como: Contexto Navegacional[21].
3. Contexto: Un componente proporciona información sobre el contexto de localización de la vista actual del usuario.
4. Problema: La falta de un esquema mental por parte del usuario cuando navega a través de las páginas de hipertexto es causa reconocida del 'síndrome de perdido en el hiperespacio'. Para evitar este problema y mejorar la facilidad de uso de la interfaz es necesario proporcionar un mecanismo que indique nuestra localización dentro de la aplicación. Existen dos fuerzas asociadas a este problema:
   (a) El camino utilizado para alcanzar los componentes de información no tiene por qué ser conocido de antemano.
   (b) El componente de localización debería ser lo más independiente posible del resto de la aplicación.
5. Solución: Implementar un componente de localización y mecanismos mediante los cuales un cambio en la vista que el usuario tiene del sistema produzca una propagación del cambio que reestablezca la consistencia entre la vista y el componente de localización.
6. Implementación: OO-$\mathcal{H}$Method propone dos posibilidades: la más sencilla consiste en utilizar, directamente asociada con cada conjunto de páginas relacionadas, una estructura de tipo cabecera-pie que, a través de sus distintos estilos y/o información textual, identifique la zona de la aplicación en la que nos encontramos. Una aproximación más general, sin embargo, implica una adaptación del patrón Observer[8] a los entornos hipermediales.

En nuestro ejemplo, además del patrón anterior, se ha aplicado el Patrón de Error[2]. Esto ha provocado la aparición de una nueva página de tipo tStruct, así como un conjunto de relaciones de dependencia para informar a las páginas donde se puede producir un error de la necesidad de invocar a esta nueva página.

Otro posible modo de refinamiento consiste en la manipulación directa por parte del diseñador de los constructores del DPA. Así, en nuestro ejemplo el diseñador ha incluido una nueva página abstracta de tipo tWindow. Las páginas estáticas asociadas con el punto de entrada de la aplicación, que en su generación por defecto no contenían más que un elemento de tipo label sin contenido, han sido también editadas por el diseñador. Este tipo de páginas, formen parte de un túnel completo de entrada/salida a la aplicación o no, son especialmente sensibles a factores gráficos, y por tanto suelen requerir la edición y maquetación independiente del modelo.

Los patrones pueden causar la aparición y/o modificación de cualquier tipo de página abstracta. Para ilustrar esta afirmación, basta señalar que la aplicación de un patrón como puede ser el Patrón de Confirmación[2] causaría la inclusión de una función de usuario de tipo 'confirm' y su invocación obligatoria antes de poder realizar, o incluso solicitar, cualquier cambio en el sistema del tipo especificado. Así, el contenido de la página abstracta de función, donde se recopila la lógica de cliente, tendría que cambiar. Todos los patrones definidos en el repositorio tienen definida una materialización por defecto, y pueden ser aplicados a la interfaz de usuario a discreción del diseñador, aunque lo normal es que se sigan unas reglas básicas como son las recogidas en [21].

En OO-$\mathcal{H}$Method también podemos elegir si deseamos que las modificaciones introducidas mediante los patrones sean visibles en el diagrama o no. En nuestro ejemplo, el conjunto de páginas abstractas y de enlaces de dependencia insertados en el diagrama como consecuencia de la aplicación del 'Patrón de Localización', no proporciona información relevante, por lo que podría ocultarse. Lo mismo ocurre con el Patrón de Error. La utilización de patrones implícitos clarifican el esquema y evitan saturar al diseñador con demasiados datos irrelevantes. Conviene resaltar que, en el ámbito de las interfaces hipermediales, y por tanto también en OO-$\mathcal{H}$Method, la implementación de dos o más mecanismos para cubrir un mismo problema es un rasgo no sólo permitido sino común. Por ejemplo, el Patrón de Observación de la Navegación [2] suele ser capturado en la interfaz mediante la aparición simultánea del botón de 'vuelta atrás' (back) y del de 'recarga' (reload).

Como ejemplo de página abstracta la estructura interna de 'ChatList', generada tras la aplicación de los refinamientos, quedaría como sigue:

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

```
<!DOCTYPE tStruct SYSTEM "tStruct.dtd" encoding="UTF-8">
<tStruct>
    <label style="" text="List of available chats" />
    <link name="error" type="automatic" show="new"
        pointsTo="tStruct" dest="errorPage">
    </link>
    <link name="head" type="automatic" show="here"
        pointsTo="tStruct" dest="head">
    </link>
    <collection format="ulist" style="schatlist">
        <object type="chat">
            <attrib name="nameChat" type="string">
            </attrib>
            <call event="onClick" function="validate"/>
        </object>
    </collection>
    <link name="foot" type="automatic" show="here"
        pointsTo="tStruct" dest="foot">
    </link>
</tStruct>
```

Una vez realizados los refinamientos, el siguiente paso es generar la interfaz mediante la aplicación de tareas y técnicas de interacción, que serán introducidas en la siguiente sección. Esta etapa ya es dependiente del entorno en el que se pretenda poner en funcionamiento dicha interfaz.

5 Implementación del DPA

Se define una Tarea de Interacción como un mecanismo que agrupa, de acuerdo a la acción que debe ser realizada, un conjunto de elementos físicos que colaboran en la cobertura de dicha acción. Estos elementos pueden haber sido capturados en cualquiera de los diagramas, o pueden formar parte de cualquiera de los patrones aplicados a esos diagramas. Por otra parte, definimos una 'Técnica de Interacción' como cada una de las posibilidades de materialización que tiene una Tarea de Interacción. Las técnicas de interacción pueden ser asociadas a una estrategia y/o entorno de programación concreto. La complejidad de la transformación entre las páginas abstractas en el DPA y los constructores finales [15] quedan fuera del ámbito de este artículo. En las Fig. 3 a 5 se muestra la interfaz generada a partir del DPA de la Fig. 2.

El proceso es como sigue: en primer lugar, la herramienta de generación localiza la página abstracta que corresponde al punto de entrada de la aplicación, así como sus posibles enlaces a páginas estáticas añadidas manualmente por el diseñador(ver Fig. 3). Se puede observar cómo todas las páginas del diagrama tienen las mismas páginas de cabecera y pie asociadas, lo cual dota a la interfaz de un contexto visual común. Cuando el usuario activa el enlace 'View Chat List' que, recordemos, estaba definido como un enlace con el par atributo-valor 'show = new' en la página de tipo tStruct asociada, la herramienta de generación rellena dicha página abstracta 'ChatList' con los objetos disponibles en la aplicación, y la página física de la Fig. 4 es generada como resultado.

Cuando el usuario decide activar el enlace 'View Message', se realiza la materialización de la página abstracta de tipo tWindow. La plantilla tWindow tiene un comportamiento bastante distinto al de tStruct: en lugar de rellenarse con objetos, esta plantilla define las características generales de las dos vistas simultáneamente activas que el usuario va a tener a partir de entonces del sistema. Así se definen dos enlaces cuyo tipo será 'automatic', es decir, activables por la propia interfaz sin necesidad de interacción con el usuario. Estos enlaces son los encargados de cargar las dos vistas simultáneas de nuestro ejemplo: el conjunto de mensajes enviados al sistema hasta el momento (una nueva página de tipo tStruct) y el formulario con los campos requeridos para añadir nuevos mensajes al foro. Ya comentamos en la sección anterior que la inclusión de la estructura tWindow era un refinamiento del diagrama. Si no hubiera existido, la implementación por defecto (a partir del DAN), habría seguido la política de 'una sola vista activa', esto es, se habría generado una pantalla con los mensajes disponibles y un enlace activable por el usuario que le llevara a otra página, de tipo tForm, donde el usuario pudiese introducir los parámetros. Como el servicio 'Reply Message' devuelve un booleano especificando el éxito o fracaso de la acción, una página conteniendo la confirmación del éxito de la operación es generada de manera transparente al

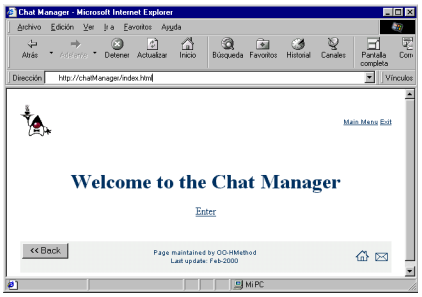


Figure3. Punto de Entrada al Gestor de Foros

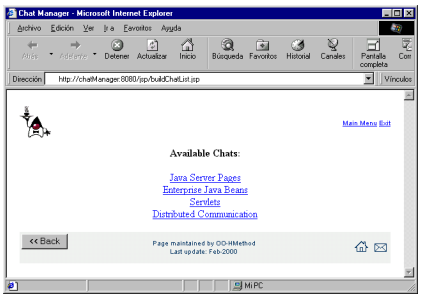
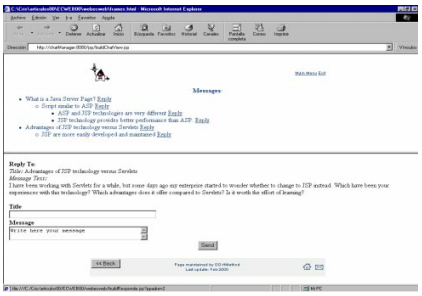


Figure4. Lista de Grupos de Discusión disponibles



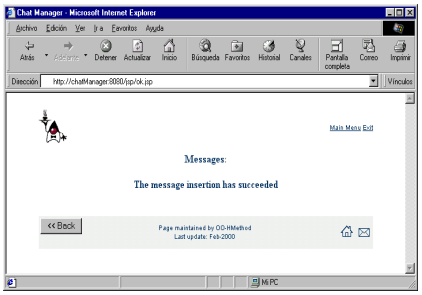


Figure5. Adición de mensajes al Foro

diseñador (see Fig. 5).

La aplicación ejemplo ha sido desarrollada utilizando Java Server Pages y un componente de tipo Bean [23] como entorno de implementación en servidor, y HTML como entorno de implementación en cliente.

6 Comparación con trabajos relacionados

La mayoría de los métodos estudiados hasta el momento comparten muchos de los conceptos que han sido identificados como relevantes en el ámbito de diseño de aplicaciones hipermediales. Tal es el caso de las colecciones, que bien con ese u otro nombre aparecen en métodos tan diversos como [9, 12, 22, 7, 1]. OO-$\mathcal{H}$Method se nutre de toda esta experiencia, y así también captura estos conceptos, pero además considera explícitamente los modelos de interacción del usuario con la lógica de la aplicación, alejándose de las aproximaciones que sólo se centran en la navegación y visualización de datos.

Por otro lado, la importancia de utilizar técnicas de modelado conceptual en el contexto Web ha sido ampliamente discutida por la comunidad científica. Sin embargo, muchas aplicaciones comerciales existentes en la actualidad se centran en el espacio de la solución. IDC's o ASP's de Microsoft, o Cold Fusion de Allaire son algunos ejemplos. Estas aproximaciones, si bien son muy flexibles, obligan al diseñador a realizar actividades en las que se tiene una alta probabilidad de cometer errores (necesidad de utilizar exactamente los mismos nombres de campos que existen en base de datos, manejo de manera explícita el enlazado de las páginas, etc). OO-$\mathcal{H}$Method, por ser un método de modelado conceptual, se aleja de estas aproximaciones y define una serie de constructores y plantillas [7, 16, 22] desde un punto de vista declarativo. Nuestro concepto de plantilla comparte varios rasgos con el utilizado en otros modelos estudiados hasta el momento[1, 6, 7, 14, 22]: en concreto, todos ellos asumen implícitamente la existencia de un esquema de visualización de página, y de algún modo especifican los datos que se deben mostrar en ese esquema. Pero hay rasgos que diferencian nuestro trabajo de otras aproximaciones: al contrario que en [1], OO-$\mathcal{H}$Method proporciona un esquema que es totalmente independiente de los detalles finales de implementación, como puede ser el carácter estático/dinámico de las páginas. Nuestra noción de plantillas va también más allá de las 'Plantillas Constructivas' presentadas en [16]. Aunque el concepto del que

parten ambas aproximaciones es similar, el hecho de que en OO-$\mathcal{H}$Method la estructura base del DPA (1) derive del DAN y (2) sea modificada principalmente mediante un conjunto de patrones facilita su diseño y modificación. Además, la definición del conjunto de DTD's que definen los distintos tipos de plantilla estandariza su estructura y apariencia, facilitando su uso. La separación de los aspectos que influyen en la interfaz final (estructura, contenido, presentación y comportamiento en cliente) mediante la taxonomía de plantillas de OO-$\mathcal{H}$Method contribuye a la mantenibilidad del código generado. También permite la captura de aspectos de interacción complejos como son la funcionalidad de cliente, la invocación de lógica o la existencia de vistas simultáneas del sistema.

Varios métodos estudiados [1, 6, 14] utilizan lenguajes de consulta para especificar la definición de la interfaz. Esta aproximación es muy potente, sobre todo en términos de rápido prototipado e integración de fuentes de información heterogéneas. Frente a esto, OO-$\mathcal{H}$Method propone un marco menos confuso en el que reflexionar sobre rasgos como el acceso a la funcionalidad o caminos de navegación más adecuados. Así, OO-$\mathcal{H}$Method ha optado a la hora de integrar datos y estructura por una aproximación basada en filtros asociados a constructores de navegación. Para definir la sintaxis de estos filtros OO-$\mathcal{H}$Method se ha basado en OCL [24].

7 Conclusiones

En este artículo hemos presentado el núcleo de OO-$\mathcal{H}$Method, una extensión a los métodos de modelado conceptual OO para la especificación de interfaces hipermediales. OO-$\mathcal{H}$Method parte de un diagrama de clases de dominio, y a partir de él define dos nuevos diagramas: el Diagrama de Acceso Navegacional (DAN) y el Diagrama de Presentación Abstracta (DPA). El DAN captura las estructuras de acceso y navegación a través de los datos para cada tipo de usuario. La personalización de la interfaz se consigue mediante la elaboración de distintos DAN's, uno para cada tipo de usuario. A partir de él se puede generar de forma automática la estructura básica del Diagrama de Presentación Abstracta (DPA). Esta estructura es útil para generar prototipos operativos. A fin de ayudar al diseñador a refinar los diagramas, OO-$\mathcal{H}$Method incluye un Catálogo de Patrones de Interfaz Hipermedial, que contribuyen además al incremento de la facilidad de uso y la calidad de la interfaz generada.

La interacción usuario-sistema, tratada de forma superficial en otros modelos, forma parte integrante de OO-$\mathcal{H}$Method: los servicios ofrecidos por la capa de lógica para cada tipo de usuario son manipulados por el diseñador desde el inicio del método, y factores como la introducción de parámetros y la recogida de los resultados de ejecución son tenidos explícitamente en cuenta.

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

1. Presentación de las fases, diagramas y constructores de OO-$\mathcal{H}$Method.
2. Uso de un Catálogo de Patrones de Interfaz para la captura de características relevantes para la calidad del modelo de interacción usuario-sistema.
3. Presentación de una taxonomía de plantillas, definidas en XML, que separan los diferentes aspectos relacionados con la generación de las páginas de interfaz.
4. Marco general de transformaciones básicas entre el DAN y el DPA.

References

[1] P. Atzeni, G. Mecca, and P. Merialdo. Design and Maintenance of Data-Intensive Web Sites. In *Advances in Database Technology - EDTB'98*, pages 436–449, 03 1998.

[2] C. Cachero. The OO-$\mathcal{H}$Method Pattern Catalog. Technical report, Universidad de Alicante, 12 1999.

[3] C. Cachero, J. Gómez, and O. Pastor. Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO-HMethod Presentation Abstract Model. In *EC-Web 2000 (to appear). 1^{st} International Conference on Electronic Commerce and Web Technologies*. Springer-Verlag. Lecture Notes in Computer Science, 09 2000.

[4] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. In *Position Paper, Web Engineering Workshop, WWW9*, 05 2000.

[5] eXtensible Markup Language (XML). http://www.w3.org/XML/.

[6] F. M. Fernández, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the Boat with Strudel: Experiences with a Web-Site Management System. In *Proceedings of ACM SIGMOD International conference on Management of data*, pages 414–425, 10 1998.

[7] P. Fraternali and P. Paolini. A Conceptual Model and a Tool Environment for Developing more Scalable, Dynamic, and Customizable Web Applications. In *Advances in Database Technology - EDBT'98*, pages 421–435, 1998.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[9] F. Garzotto and P. Paolini. HDM A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems (TOIS)*, 11(1):1–26, 01 1993.

[10] D.M. Germán and D.D. Cowan. Three Hypermedia Design Patterns. In *Proceedings of the HT99 Workshop on Hypermedia Development: Design Patterns in Hypermedia*, 02 1999.

[11] J. Gómez, C. Cachero, and O. Pastor. Extending a Conceptual Modelling Approach to Web Application Design. In *CAiSE '00 (to appear). 12^{th} International Conference on Advanced Information Systems*. Springer-Verlag. Lecture Notes in Computer Science, 06 2000.

[12] T. Isakowitz, E. A. Stohr, and V. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *CACM: Communications of the ACM.*, pages 34–44, 08 1995.

[13] S. McGrath. *XML by Example. Building e-commerce Applications*. Prentice Hall, 1998.

[14] G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The ARANEUS Guide to Web-Site Development. Technical report, Universidad de Roma, 03 1999.

[15] P. Molina. Especificación de la Interfaz de Usuario en OO-Method. Technical report, Universidad Politécnica de Valencia, 1998.

[16] M. Nanard, J. Nanard, and P. Kahn. Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. In *HYPERTEXT '98. Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space—structure in hypermedia systems*, pages 11–20, 1998.

[17] OMG Unified Modeling Language Specification Version 1.3. http://www.omg.org/cgi-bin/doc?ad/99-06-08.pdf, 06 1999.

[18] P. Paolini and F. Garzotto. Design Patterns for WWW Hypermedia: Problems and Proposals. In *Proceedings of the HT99 Workshop on Hypermedia Development: Design Patterns in Hypermedia*, 02 1999.

[19] O. Pastor, E. Insfrán, V. Pelechano, J. Romero, and J. Merseguer. OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. In *CAiSE '97. International Conference on Advanced Information Systems*, pages 145–158, 1997.

[20] O. Pastor, V. Pelechano, E. Insfrán, and J. Gómez. From Object Oriented Conceptual Modeling to Automated Programming in Java. In *ER '98. International Conference on the Entity Relationship Approach*, pages 183–196, 1998.

[21] G. Rossi, D. Schwabe, and A. Garrido. Design Reuse in Hypermedia Applications Development. In *Proceedings of the eight ACM conference on HYPERTEXT '97*, pages 57–66, 1997.

[22] D. Schwabe, G. Rossi, and D. J. Barbosa. Systematic Hypermedia Application Design with OOHDM. In *Proceedings of the the seventh ACM conference on HYPERTEXT '96*, page 166, 1996.

[23] The source for java technology. http://java.sun.com.

[24] Jos Warmer and Anneke Kleppe. *The Object Constraint Language. Precise Modeling with UML*. Addison-Wesley, 1998.