Contenido en la *Web*: un modelo de reutilización

Arantza Irastorza, Arturo Jaime, Oscar Díaz [1]

Grupo EKIN

Dpto. Lenguajes y Sistemas Informáticos
Universidad del País Vasco / Euskal Herriko Unibertsitatea
Apdo. 649 – 20.080 Donostia
{jipirgoa, jipjaela, jipdigao}@si.ehu.es

Resumen. El continuo crecimiento en tamaño y complejidad de las aplicaciones *Web* precisa de propuestas rigurosas y sistemáticas que permitan afrontar las demandas tanto en el desarrollo como en el mantenimiento de estos sistemas. Una de las estrategias más difundidas en la ingeniería del software tradicional es la reutilización. Este trabajo aborda la reutilización en aplicaciones *Web*. Para ello introduce la noción de "unidades de datos" que considera los requisitos de las aplicaciones *Web* basadas en el contenido. Una unidad de datos (UD) es un documento XML que aglutina junto con el contenido, los aspectos de presentación, navegación y traza de forma separada. Una unidad de datos puede ser reutilizada desde otra unidad de datos mediante un mecanismo de importación. También es posible la adaptación de una unidad de datos, limitada a las opciones previstas por su desarrollador. Se introduce el lenguaje de marcas AtariX que permite describir todas estas cuestiones.

1 Introducción

El desarrollo actual de aplicaciones *Web* se encuentra con un serio cuello de botella debido a la diferencia de nivel entre las herramientas de implementación disponibles y los requisitos de las aplicaciones. Se está ejerciendo una gran presión sobre los desarrolladores para que "codifiquen y publiquen". La formación de estos desarrolladores suele ser bastante diversa, siendo habitual que no tengan ninguna experiencia en ingeniería de software. Se guían por las características de las herramientas de desarrollo y los constructores de los lenguajes. Este estilo de desarrollo libre puede conducir a *"emplear estrategias ad hoc o estilo hacker, carentes de rigor, de técnicas sistemáticas, de buenas metodologías, y sin garantía de calidad"* [5]. Estas prácticas pueden ser perniciosas de cara a afrontar el mantenimiento de estas aplicaciones. Es preciso traer al campo de las aplicaciones *Web* las prácticas de la ingeniería de software convencional, acomodándolas a su propia idiosincrasia. Las técnicas de reutilización son una de estas prácticas. Krueger [7] define la reutilización de software como *"el proceso de crear sistemas de software a partir de software existente en vez de construirlo desde cero"*.

Las aplicaciones *Web* han ido evolucionando desde centrarse puramente en los contenidos (*e-brochures*, *e-marketing*) a ofrecer distintos tipos de servicios (*e-commerce*, *e-business*). Este trabajo se centra exclusivamente en los aspectos del contenido para los que se propone un modelo de reutilización. El desarrollo de software es una tarea compleja que precisa gran cantidad de tiempo y dinero. Por ello es recomendable que el desarrollador reutilice el máximo de trabajo realizado y probado con anterioridad. El desarrollo de aplicaciones para la *Web* no es una excepción, y si cabe, tiene mayor necesidad, dada la gran rapidez con la que evoluciona este tipo de aplicaciones. Continuamente se crean nuevos sitios *Web* que ofrecen gran cantidad de información. Estos sitios cambian a menudo su imagen, ofrecen nuevas posibilidades de navegación, incluyen entre su información referencias a otros sitios *Web*, etc. Consideremos el ejemplo de una conferencia donde se van a celebrar varios talleres en paralelo. Es bastante usual que la organización de los talleres y de la conferencia no recaiga en las mismas personas y que por tanto, los diseñadores de los sitios *Web* correspondientes a los talleres y la conferencia tampoco coincidan. Sin embargo, sería deseable que desde el sitio *Web* de la conferencia pudiera accederse a toda la información relativa a sus talleres. Para ello los diseñadores del sitio de la conferencia tendrán que elegir entre reutilizar lo que hayan hecho los diseñadores de los sitios de los talleres o rehacerlo de acuerdo a sus pautas de presentación, etc. La última opción parece inadecuada de cara a mantener la consistencia de la información ofrecida y necesitará más tiempo para ofrecer su producto.

[1] Este trabajo ha sido financiado parcialmente por la Secretaría de Estado de Política Científica y Tecnológica, bajo el proyecto TIC1999-1048-C02-02

Para diseñar, implementar y reutilizar aplicaciones *Web* de contenido hemos definido el concepto de unidad de datos (UD).

- **Una UD se centra en los datos**, y no en los servicios que se puedan realizar desde una aplicación *Web*. Dicha unidad incluye no sólo los datos propiamente dichos sino también su formato de presentación y navegación. Cuando una UD es compilada genera "un trozo" de código HTML repartido en una o varias páginas. Dichas páginas permiten visualizar datos pero nunca ejecutar servicios.
- **Una UD es una unidad de diseño.** Los objetivos de una aplicación *Web* difícilmente pueden rastrearse acudiendo a una sola página. La página puede ser la unidad de implementación pero difícilmente será la unidad de diseño. Cada objetivo del diseño normalmente queda plasmado en distintas páginas. Por ejemplo, supóngase un sistema tipo *Amazon*. Una de las funciones del sistema es la localización de un libro. Claramente, los datos necesarios para soportar esta función se localizan en un conjunto de páginas HTML. Precisamente, una UD se concibe como una unidad lógica que aglutina los datos de un conjunto de páginas físicas. La página HTML se concibe como una unidad de implementación mientras que una UD intenta ser una unidad de diseño al soportar un requisito de información del usuario (por ejemplo, localizar un libro). Por tanto una UD es la base de la implementación de una aplicación *Web* de contenido pura.
- **Una UD soporta un modelo de reutilización *de caja blanca***. Sametinger distingue entre reutilización *de caja negra* y *de caja blanca* [9]. En el primer tipo no es posible modificar el software a reutilizar, debiéndose reutilizar tal cual está. Por el contrario, el segundo tipo permite al reutilizador ver las 'interioridades' del software e incluso adaptarlo a sus propias necesidades. Ente ambos extremos se encuentran otros tipos de reutilización denominados *de caja de cristal* y *de caja gris*. Nuestro modelo de reutilización incorpora un mecanismo de *caja blanca*, aunque como ya mostraremos podría extenderse para implementar un mecanismo *de caja gris*, que restringe las posibilidades de adaptación.

La razón por la que los datos desempeñan un papel nuclear en las aplicaciones *Web* se basa en la propia idiosincrasia de la *Web*: "salvo en aplicaciones de Internet desarrolladas para proporcionar servicios, los sitios *Web* se parecen más a revistas o folletos que a aplicaciones software (...) están enfocados hacia la comunicación, a diferencia de las aplicaciones software tradicionales" [8].

El resto del artículo tiene la siguiente estructura. Primero se introduce el concepto de *unidad de datos UD*. A continuación se presenta un modelo de reutilización para las unidades de datos. El artículo concluye con un resumen de trabajos relacionados y una sección de conclusiones.

2 Definición de las unidades de datos

2.1 Contenido y estructura

A diferencia de la estructura de datos relacional, que se caracteriza por ser regular y con atributos atómicos, una UD es normalmente semi-estructurada con atributos complejos, donde además el orden de los elementos es significativo. Por ejemplo, en un sitio *Web* de una conferencia los datos de los asistentes podrían seguir el siguiente patrón: para cada asistente se recopila la misma información, donde el grano de los datos es atómico y el orden en el que se organizan no lleva implícito ningún significado (estructura relacional). Por contra, la información introductoria sobre la conferencia, como la presentación, temas de interés, diferentes plazos e instrucciones para los autores, tendrían una estructura irregular. De cara a acomodar las particularidades de cada conferencia, estos aspectos no siguen una estructura común. Por ejemplo, una conferencia concreta podría incluir un viaje en barco y proporcionar guías estrictas dirigidas a los autores sobre la confección de sus artículos, mientras que otras conferencias podrían no incluir ninguno de dichos aspectos.

Para definir las UDs utilizamos el lenguaje de marcas AtariX, que es una extensión de XML. La especificación XML [11] tiene el potencial de hacer efectiva la integración de datos estructurados y semi-estructurados y está siendo ampliamente adoptada como estándar de representación e intercambio de información. El constructor básico de XML es el elemento, esto es, el texto delimitado por etiquetas correspondientes entre sí, tales como <*persona*> y </*persona*>. Dentro de un elemento se sitúa el contenido. Dicho contenido puede ser texto simple (por ejemplo, el nombre de una persona), otros subelementos (como <*nombre*> </*nombre*>, <*apellido*> </*apellido*>) o una mezcla de ambos. Además,

los elementos pueden caracterizarse por una serie de atributos, por ejemplo, *<persona tema="bd">Lorena</persona>* que serán utilizados en el procesamiento del documento que contiene dichos elementos. Todos los elementos de un documento XML se organizan en una jerarquía, con un solo elemento raíz.

La figura 1 ilustra el contenido y estructura de la UD *workshop* que recoge los datos que pertenecen a un taller de una conferencia. La raíz de este documento es el elemento WORKSHOP del que cuelgan NAME, ORGANIZER, etc. El elemento ACCEPTED_PAPER a su vez contiene distintos sub-elementos: AUTHOR, TITLE, AREA, etc. Esta jerarquía de elementos conforman la estructura de la UD. Su contenido se incluye dentro de cada elemento ("*Hong Kong*", "*Database*", etc.).

```
<?xml version="1.0"?>
 <?atarix_navigation_document   href="workshop.navigation.xml"   type="text/xml"?>
 <?atarix_presentation_document href="workshop.presentation.xml" type="text/xml"?>
 <?atarix_trace_document        href="workshop.tracing.xml"      type="text/xml"?>
 <WORKSHOP>
    <NAME> SEMANTIC ISSUES IN e-COMMERCE SYSTEMS </NAME>
    <ORGANIZER> IFIP working group 2.6(database) </ORGANIZER>
    <PLACE> Hong Kong </PLACE>
    <AREA> Database </AREA>
    <DATE> April 25-28, 2001 </DATE>
    <DEADLINE> December 8, 2000 </DEADLINE>
    <CALL_FOR_PAPER>
        <NAME>Deadline for submission of abstract</NAME>
        <DATE>30/11/2000</DATE>
    </CALL_FOR_PAPER>
    ...
    <ACCEPTED_PAPER>
       <AUTHOR>Kenneth R. Jacobs</AUTHOR>
       <TITLE>Innovation in Database management: Computer Science vs Engineering</TITLE>
       <AREA> Data Management </AREA>
       <DATE> April 25,2000 </DATE>
       <ROOM> 1 </ROOM>
       <TIME_SLOT> 9:00-10:00 </TIME_SLOT>
    </ACCEPTED_PAPER>
    ...
 </WORKSHOP>
```

Figura 1. La UD *workshop*. Las instrucciones de proceso se utilizan para indicar los aspectos de presentación, navegación y traza.

Una UD no tiene que corresponder necesariamente con un tipo de entidad, tal y como se concibe por quienes trabajan en bases de datos. Tampoco es un tipo abstracto de datos ya que no son las operaciones, sino su contenido y su estructura lo que caracteriza a la UD. Así, dos UDs con la misma estructura pero diferente contenido son UDs diferentes. Obsérvese que el contenido es estático y no refleja el estado de la aplicación *Web*. Aunque puede ser generado dinámicamente, no puede modificarse en el transcurso de una sesión.

2.2 Presentación, navegación y traza

A diferencia de HTML, un documento XML no proporciona ninguna instrucción sobre cómo visualizarlo. Dicha información puede incluirse de forma separada en una hoja de estilo. Las hojas de estilo se escriben en un lenguaje de especificación llamado XSL y se utilizan para transformar datos XML en HTML (o cualquier otra descripción). Siguiendo con esta idea hemos tratado de identificar diferentes dimensiones ortogonales que surgen durante la implementación de una aplicación *Web*, de manera que cada dimensión se describa separadamente. Distinguimos tres aspectos diferentes: cómo se presentan al usuario los elementos que conforman la UD, cómo se navega a través de ellos y cómo se trazan[2]. La UD también incluye la descripción de esas tres dimensiones.

Como podemos apreciar, la definición de contenido y estructura de la figura 1 no se diferencia de cualquier otro documento XML. Sin embargo, se han introducido nuevas instrucciones de proceso (*atarix_navigation_document*, *atarix_presentation_document* y *atarix_trace_document*) que sirven para enlazar la estructura y su contenido con las dimensiones mencionadas de navegación, presentación y traza. En este caso estas dimensiones están descritas en los documentos *workshop.navigation.xml*,

[2] Utilizaremos *trazar* como traducción del verbo inglés *to trace*, con el significado de hacer un seguimiento de los datos que el usuario ha ido visualizando en la aplicación *Web*.

workshop.presentation.xml y *workshop.tracing.xml*, definidos en AtariX. La figura 2 muestra el resultado de aplicar las instrucciones de proceso al contenido de la UD *workshop*. En la parte superior de la página se muestran los datos relativos al taller y un enlace para mostrar los artículos. Una vez que el usuario ha pinchado en éste, se muestra en la parte inferior izquierda un índice con los autores de los artículos. Suponiendo que el usuario ha seleccionado el autor *K. Jacobs*, se muestra en la parte inferior derecha el contenido de su artículo *Innovation in Database Management: ....*

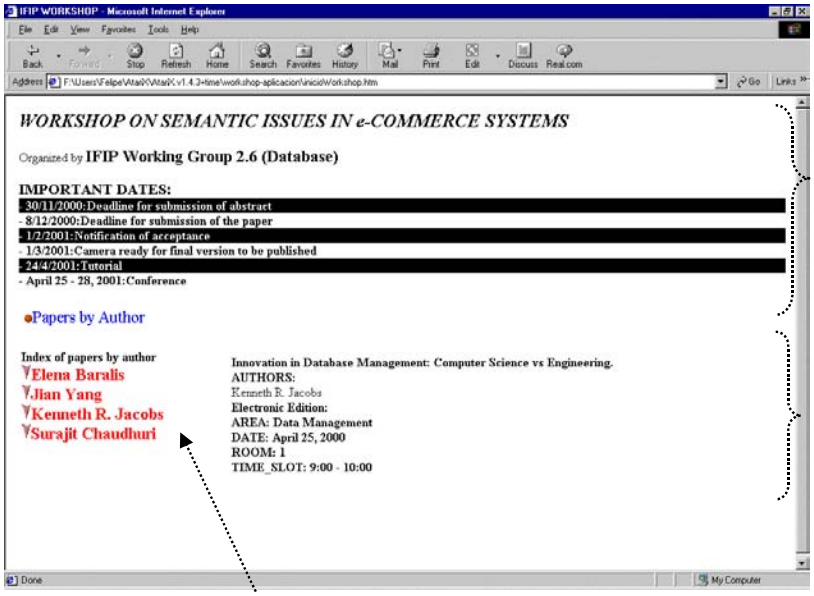

Figura 2. Página general del sitio *Web* de un taller.

A continuación describiremos con más detalle cómo se especifica en AtariX el aspecto de navegación. Una explicación pormenorizada de éste y el resto de los aspectos se puede encontrar en [2].

La navegación se refiere al modo y orden en el que se pueden visitar los diferentes elementos de una UD. Se indica la topología de los enlaces dentro de la UD. Consideremos el ejemplo del *workshop*, ¿conviene presentar juntos todos los datos sobre el *workshop*?. Para evitar mostrar páginas sobrecargadas de información, el diseñador puede decidir presentar inicialmente los elementos del primer nivel de la jerarquía que comienza en la etiqueta WORKSHOP (como NAME, ORGANIZER, etc.), y situar los ACCEPTED_PAPERs en una página diferente. Si hubiera un solo ACCEPTED_PAPER, la navegación sería sencilla. Se necesitaría un simple enlace desde la página inicial hacia la página que incluye los datos del ACCEPTED_PAPER. Pero, ¿qué ocurre si hay cien artículos?. La navegación hace frente a estos problemas.

AtariX introduce el constructor LINK para especificar estas cuestiones. Supongamos que el elemento WORKSHOP se visualiza en un lugar de la página mientras que los elementos ACCEPTED_PAPERs se muestran en la misma página o en otra. Si se desea un enlace hipermedia que nos permita movernos de la zona de presentación de WORKSHOP a la de ACCEPTED_PAPERs lo especificaremos de la siguiente manera:

```
<LINK title = "TO_ACCEPTED_PAPERS" from = "/WORKSHOP" to = "ACCEPTED_PAPER"> ... </LINK>
```

Este elemento LINK consta de un conjunto de atributos que describen (1) la etiqueta del enlace que aparecerá en la pantalla (atributo *title*); (2) el origen del enlace (atributo *from*) que indica cuándo estará disponible el enlace (en este caso cuando se muestre el elemento WORKSHOP); y (3) el destino del enlace (atributo *to*) que recoge el elemento a ser mostrado cuando se siga ese camino (en este caso ACCEPTED_PAPER). Los diferentes tipos de elementos de la estructura de la UD se conciben como posibles puntos sobre los que establecer orígenes y destinos. Estos elementos se direccionan utilizando el estándar de caminos XPath de W3C [12]. XPath concibe un documento XML como un árbol de nodos, y emplea una notación similar a la del sistema de ficheros de UNIX.

De la especificación anterior se desprende la existencia de un enlace desde WORKSHOP a ACCEPTED_PAPER. Sin embargo, si el número de nodos destino es alto, el diseñador tiene que establecer cómo se presentarán dichos nodos. Para este fin, AtariX proporciona índices, filtros y *scrolls* [1]. Así la definición anterior se puede extender con un índice de la siguiente manera:

```
<LINK title = "TO_ACCEPTED_PAPERS" from = "/WORKSHOP" to = "ACCEPTED_PAPER">
    <INDEX order="1" title="Index of papers by author" couplingMode=" embedOnRequest">
          <INDEX_PROPERTY propertyName="AUTHOR"/>
    </INDEX>
    <CONTENT order="2" couplingMode="embedOnRequest"/>
</LINK>
```

El resultado se muestra en la figura 2. En lugar de mostrar el conjunto completo de artículos a la vez, un índice proporciona una selección ordenada. En este caso se usa un índice por AUTHOR donde los artículos se van obteniendo según se van pulsando los enlaces del índice.

3 Reutilización de las unidades de datos

Volviendo a nuestro ejemplo inicial, donde considerábamos el caso de una conferencia con talleres paralelos que quería introducir en su sitio *Web* información relativa a éstos, veíamos que el diseñador del sitio *Web* de la conferencia tenía dos posibilidades: reutilizar lo que ya estaba desarrollado para los sitios *Web* de los talleres o rehacerlo de acuerdo a sus propias pautas de presentación, etc. Esta última opción no parece del todo adecuada si tenemos en cuenta que una de las características más importantes de las aplicaciones *Web* es la presión del mercado para obtener desarrollos rápidos.

En caso de reutilizar lo ya hecho, la solución más simple y directa sería incluir una instrucción *href* en todas las páginas HTML del sitio *Web* de la conferencia. Sin embargo, dado que este tipo de reutilización está basado en las páginas, estamos forzados a reutilizar no sólo los datos mostrados en la página, sino también su presentación y el resto de los aspectos. Teniendo en cuenta que los sitios desarrollados de manera independiente por diferentes personas generalmente no siguen estándares comunes (difieren los fondos de las páginas, los colores utilizados, las recomendaciones seguidas para construir los iconos, ...), el sitio integrado (en nuestro caso *conferencia* + *taller*) sufrirá de la falta de una apariencia común. Además, con esta aproximación el diseñador estará obligado a reutilizar todo el contenido de la página referenciada, aunque estuviera interesado en reutilizar únicamente un subconjunto del mismo (por ejemplo, el nombre del taller o sus artículos aceptados). La reutilización de UD aborda estos problemas.

El proceso de reutilización consta de dos partes (a) describir qué se puede reutilizar, y (b) describir qué se va a reutilizar. En nuestro modelo, la reutilización se define en términos de la estructura, es decir, los elementos de la UD (por ejemplo, WORKSHOP, ACCEPTED_PAPER, PLACE, DATE), por tanto la primera cuestión se resuelve directamente con la propia descripción de la estructura y el contenido. Respecto al segundo aspecto, mediante la etiqueta <atarix:IMPORT> se indica la intención de reutilizar ciertos elementos de una UD externa, de forma que reutilizar un elemento implica reutilizar los sub-elementos que lo tienen como padre. Además, dicha reutilización incluye no sólo el contenido, sino también la presentación, navegación y traza asociada a este elemento. En el ejemplo de la figura 3 se muestra cómo la UD *conference*, además de sus elementos propios (como HEADER y TITLE), describe la reutilización de otros dos elementos más, provenientes de la UD *workshop*: WORKSHOP_NAME y WORKSHOP_PAPER.

La etiqueta <atarix:IMPORT> se caracteriza por los siguientes atributos:

- *From*. Hace referencia al elemento que se reutiliza, indicando en qué UD se encuentra y cuál es el camino de la jerarquía hasta llegar a dicho elemento. Por ejemplo, con *workshop.xml#WORKSHOP/ACCEPTED_PAPER* se indica que se reutilizará el elemento ACCEPTED_PAPER de la UD localizada en *workshop.xml*. Además como este elemento tiene varios sub-elementos (AUTHOR, TITLE, AREA, etc.), de manera implícita éstos también serán reutilizados.
- *PageLayout*. Referencia el fichero XSL donde se indica la presentación general de los elementos importados. Si en lugar de un nombre de fichero se indica la palabra THIS significa que se utilizará el fichero de formato definido para la UD receptora (en nuestro ejemplo *conference.presentation.xml*). Por defecto se utiliza el formato de la UD importada (es decir, *workshop*).
- *ElementLayout*, *navigation* y *trace*. Hacen referencia a los documentos que describen cómo deberían presentarse, navegarse y trazarse los datos reutilizados. En caso de no especificar nada, se utilizarán las instrucciones indicadas en la propia UD reutilizada.

Como se indica en la figura 3, la UD *conference* reutiliza los elementos ACCEPTED_PAPER y NAME de la UD *workshop*, y los renombra como WORKSHOP_PAPER y WORKSHOP_NAME, respectivamente. Además, estos elementos seguirán las instrucciones de formato de página definidas en *conference* (porque, el valor del atributo *pageLayout* es *THIS*). Además, dado que el diseñador del sitio de la conferencia desea para ACCEPTED_PAPER otro tipo de navegación diferente de la ofrecida en el sitio

Web del taller, para el atributo *navigation* se indica el fichero *workConfNav.xml*. En éste se especifica la nueva navegación que sustituye a la descrita en *workshop.navigation.xml* de *workshop* (ver figura 1). Como no se indican valores para los atributos *elementLayout* y *trace*, la UD *conference* reutiliza las descripciones de formato (*layout*) y traza (*trace*) de *workshop*. Mostramos en la figura 4 la página inicial para el sitio de la conferencia, que incluye datos relativos a un taller. En la parte superior de la página se muestran los datos relativos a la conferencia y dos enlaces *Accomodation* y *Workshops*, una vez que el usuario ha pinchado en éste, se muestra en la parte inferior izquierda un índice de áreas en las que se clasifican los talleres. Suponiendo que el usuario ha seleccionado el área 'Database', se muestra en la parte inferior derecha el contenido del *workshop Semantic Issues in e-Commerce Systems*. Entre este contenido se presenta un enlace para los artículos. Cuando se pincha este enlace se muestra una lista de áreas en las que pueden encontrarse artículos. Una vez seleccionada el área *Data Management* se muestran sus artículos, pero de uno en uno utilizando las flechas de un *scroll*.

```
<?xml version="1.0"?>
 <?atarix_navigation_document   href="conference.navigation.xml"   type="text/xml"?>
 <?atarix_presentation_document href="conference.presentation.xml" type="text/xml"?>
 <?atarix_trace_document        href="conference.tracing.xml"      type="text/xml"?>
 <CONFERENCE>
    <HEADER> International Federation for Information Processing </HEADER>
    <HEADER> IFIP 2.6 WORKING CONFERENCE ON DATABASE SEMANTICS (DS-9) </HEADER>
    <TITLE> e-COMMERCE SYSTEMS ENGINEERING </TITLE>
    <ORGANIZER> IFIP working group 2.6(database) </ORGANIZER>
    ...
    <ASSOCIATED_WORKSHOP>
        <WORKSHOP_PAPER>
            <atarix:IMPORT from="workshop.xml#WORKSHOP/ACCEPTED_PAPER"
                           pageLayout="THIS"
                           navigation="workConfNav.xml"/>
        </WORKSHOP_PAPER>
        <WORKSHOP_NAME>
            <atarix:IMPORT from="workshop.xml#WORKSHOP/NAME"
                           pageLayout="THIS"/>
        </WORKSHOP_NAME>
        ...
    </ASSOCIATED_WORKSHOP>
    ...
 </CONFERENCE >
```

Figura 3. Reutilización de elementos de *workshop* en la UD *conference.*

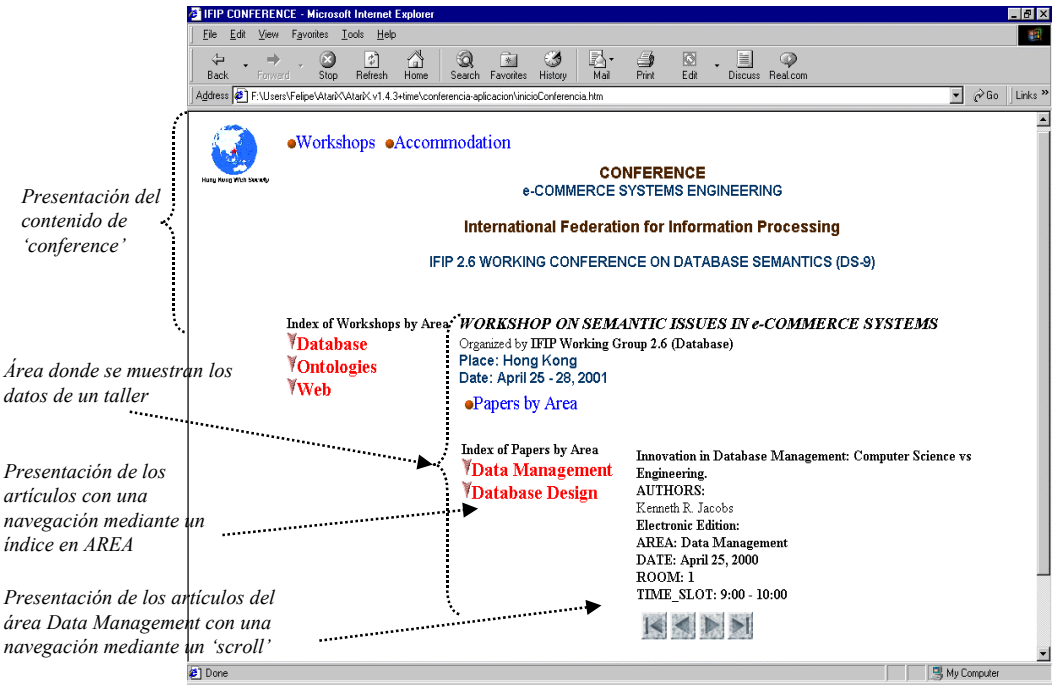

Figura 4. Página general del sitio *Web* de una conferencia.

Como puede deducirse de la descripción anterior, una UD no implica unidad de encapsulación. Una UD está formada tanto por la estructura de los elementos como por la presentación, navegación y traza.

La primera constituye el foco de atención, el medio para atraer a otros diseñadores que pueden estar interesados en reutilizar los datos. El resto son dimensiones que 'colorean' dichos datos y que permanecen visibles para que los diseñadores que vayan a reutilizar la unidad puedan cambiarlas a su gusto.

Como el ejemplo anterior ha tratado de ilustrar, las características del modelo de reutilización propuesto son:

1. La especificación de la reutilización se hace en términos de los elementos del documento XML de contenido. Es decir, en el momento de la reutilización se debe decidir e indicar la UD y sus elementos XML. Así, se puede indicar que se quiere reutilizar el elemento ACCEPTED_PAPER o AREA de *workshop*. La reutilización de los aspectos de navegación, presentación y traza está supeditada a la reutilización del elemento al cual van asociados dichos aspectos. No puede indicarse de manera independiente. Por tanto, al reutilizar el elemento AREA implícitamente también se puede reutilizar su presentación y navegación. Aunque se decida cambiar alguno de dichos aspectos, siempre será relacionándolo con AREA. No se reutiliza un índice de navegación o un estilo de presentación de forma aislada.
2. El ámbito de reutilización es un conjunto de páginas *Web*. La UD implementa una aplicación *Web*, es decir, se presenta al usuario como un conjunto de páginas *Web*. Por ello reutilizar un elemento de la UD, de hecho, significa reutilizar una parte completa de dicha aplicación, de dichas páginas. Decimos "completa" porque junto con el contenido del elemento se recogen también los formatos de presentación, navegación y traza asignados al mismo. Dado que es posible que en la UD de origen la parte reutilizada se mostrara a través de diferentes páginas *Web*, en la nueva UD también se mostrará en varias páginas *Web*.
3. Es un modelo basado en prototipos. No existen conceptos tales como clases o instancias. Cualquier UD puede ser reutilizada por otra UD. Al crear esta unidad se hace una copia de los elementos importados de la unidad reutilizada, obteniendo un objeto independiente. Esta decisión se basa nuevamente en las características del desarrollo *Web*. Como indican los autores de *WebComposition model*, *"este paradigma (la propuesta de prototipos) encaja de forma natural con el modelado de aplicaciones Web, primero porque muchas entidades Web son únicas y segundo, porque el modelo de construcción por prototipos refleja el tipo de reutilización copiar-y-modificar aplicado a menudo en el desarrollo Web"* [4].
4. Es prácticamente una reutilización *de caja blanca*, donde el diseñador de la unidad importadora puede cambiar cualquier aspecto (es decir, navegación, presentación, traza) del elemento importado, sin ninguna restricción por parte del propietario de la UD importada.

En lo que respecta a este último aspecto, actualmente estamos valorando la posibilidad de un enfoque parametrizado (*de caja gris*). Así, el propietario de la UD importada puede restringir las alternativas disponibles al diseñador que reutiliza. El autor de una UD puede predeterminar, por ejemplo, las distintas formas de navegación para uno de sus elementos. Al definir el enlace hipermedia TO_ACCEPTED_PAPERS del ejemplo de la sección anterior, puede ofrecerse la posibilidad de, por ejemplo, mostrar todo el contenido de una vez o mostrar un índice que permita ordenar la salida de información. Una posible forma de indicar esta posibilidad de elección sería utilizar una etiqueta ALTERNATIVE de la siguiente manera:

```
<LINK title = "TO_ACCEPTED_PAPERS" from = "/WORKSHOP" to = "ACCEPTED_PAPER">
   <ALTERNATIVE name="with_index">
       <INDEX order="1" title="Index of papers by author" couplingMode="embedOnRequest">
            <INDEX_PROPERTY propertyName="AUTHOR"/>
       </INDEX>
       <CONTENT order="2" couplingMode="embedOnRequest"/>
   </ALTERNATIVE>
   <ALTERNATIVE name="without_index">
       <CONTENT order="1" couplingMode="embedOnRequest"/>
   </ALTERNATIVE>
</LINK>
```

Al importar el elemento ACCEPTED_PAPER, la UD importadora está limitada a elegir únicamente una de las opciones mencionadas, por ejemplo de la siguiente manera, donde el valor del atributo *name* debe coincidir con alguna de las alternativas prefijadas por el propietario de la UD importada:

```
<LINK title = "TO_ACCEPTED_PAPERS">  <ALTERNATIVE name="with_index"/>  </LINK>
```

Si bien la parametrización es una práctica habitual en los módulos de software tradicionales, no está claro que este enfoque sea extrapolable en la *Web*. Dado que los aspectos a parametrizar están muy ligados a la presentación, es muy probable que nos movamos entre dos extremos. Para aquellas UD desarrolladas en casa siguiendo una serie de estándares conocidos, la reutilización será prácticamente

total. Sin embargo, para aquellas UD desarrolladas fuera, es razonable suponer que el desarrollador de la aplicación importadora desee total libertad a la hora de ajustar la presentación reutilizada a los cánones utilizados en dicha aplicación.

4 Otros modelos de reutilización

Esta sección examina diferentes propuestas para la reutilización. Las tres primeras, *WebComposition Model*, *Web Object Composition Model* y *Tiles Model*, están centradas básicamente en la reutilización de una parte de una página HTML. La cuarta, *Web Service User Interface Model*, se centra en la lógica de negocio. Los ejemplos mostrados en cada apartado están recogidos de las referencias mencionadas. En este trabajo se han simplificado con el objetivo de que la explicación se lo más clara y concisa posible.

4.1 WebComposition Component Model

El modelo *WebComposition* [4] está basado en un paradigma de prototipos donde la distinción entre instancias y clases no es muy precisa y un objeto (prototipo) puede heredar de cualquier otro objeto. A estos objetos en el modelo se les llama componentes. Los componentes, sus propiedades y relaciones, son descritos a través del lenguaje de marcas *WebComposition Markup Language* (WCML). Para mejorar la reutilización y la abstracción en el diseño de aplicaciones *Web*, *WebComposition* se basa en la herencia por prototipos. Se pueden definir prototipos de alto nivel que son especializados en prototipos más cercanos a la implementación. Los prototipos de alto nivel se corresponden con unidades estructurales comunes en una página (por ejemplo, una cabecera, un logotipo, etc.) mientras que los prototipos de más bajo nivel describen contenido y su presentación.

```
<component id='CLogin'>
   <property name='image' value=''/>
   <property name='version'> Version 1.122.58 </property>
</component>

<component id='CWindowsCELogin'>
   <prototype is='CLogin'/>
   <property name='image' value='tecologo256color.gif'/>
   <property name='content'>
      <table border="0" cellspacing="5"><tr>
         <td><refprop name='content' from='CLogo' prototype='CWindowsCELogin'/></td>
         <td><refprop name='content' from='CApplicationTitle'/></td>
         <td><refprop name='content' from='CForm'/></td>
      </tr></table>
      <refprop name='content' from='CCopyright'/>
   </property>
</component>
```

Figura 5. Definición de componentes en WCML.

La figura 5 muestra la definición de los componentes *CLogin* y *CWindowsCELogin*. Cada componente tiene un identificador para poder hacerle referencias y varias propiedades simples (pares nombre-valor). Por ejemplo, el componente de identificador *CLogin* está caracterizado por tener una imagen (*image*) y una versión (*version*). El componente *CWindowsCELogin*, por su parte, con la propiedad *is* indica que especializa a *CLogin*, heredando sus propiedades *version* e *image*, aunque esta última la sustituye por su propia *image*. Además *CWindowsCELogin*, por medio de la propiedad de nombre *content*, describe su implementación *Web*, es decir, cómo se mostrará utilizando HTML. Es aquí donde deja parte de la descripción a otros componentes, es decir, reutiliza las descripciones de *CLogo*, *CApplicationTitle*, etc.

4.2 Web Object Composition Model

WOCM [6] es un modelo de metadatos formal para el diseño de la estructura y el contenido de aplicaciones *Web*. Al contrario que en el caso anterior, en éste sí se hace una separación clara entre instancias y clases. Se definen dos tipos de constructores. Las clases *simplexon* sirven para definir el contenido y su representación, definen tipos abstractos de datos. Las clases *complexon* definen la estructura, el formato, de la aplicación *Web*, y son contenedores de los primeros. Un componente constituye un tipo especial de estos últimos y representa una de las partes físicas de una aplicación *Web*

(por ejemplo, una página Web). En la figura 6 se muestran las definiciones de *i1EmployeeHtml* e *i1Contact*, instancias del *complexon Employee* y del *simplexon Contact*, respectivamente. *i2EmployeeHtml* es una instancia de componente que también es instancia del *complexon Employee* y en su propiedad *resource* señala el identificador de la dirección donde puede ser localizado.

```
<lr:Employee rdf:ID="i1EmployeeHtml">
   <xwmf:hasPart>
      <rdf:Seq>
         <rdf:li rdf:resource="#htmlDocumentBegin"/>
         <rdf:li rdf:resource="#i1Contact"/>
         <rdf:li rdf:resource="#htmlDocumentEnd"/>
      </rdf:Seq>
   </xwmf:hasPart>
</lr:Employee>

<lr:HtmlContact rdf:ID="i1Contact">
   <lr:name>Reinhold Klapsing</lr:name>
   <lr:phone>+49 (201) 183 4078</lr:phone>
   <lr:fax>+49 (201) 183 4073</lr:fax>
   <lr:email> Reinhold.Klapsing@uni-essen.de</lr:email>
</lr:HtmlContact>

<lr:Employee rdf:ID="i2EmployeeHtml">
   <xwmf:isComponent rdf:resource= "http://.../Klapsing.html"/>
   <xwmf:isView> HTML </xwmf:isView>
   <xwmf:hasPart>
      ...
   </xwmf:hasPart>
</lr:Employee>
```

Figura 6. Definición de estructuras en WOCM.

La reutilización se define a través de la herencia. Todas las instancias de un *simplexon*, o de un *complexon*, heredarán el formato de presentación definido en su clase. Así, los datos de *i1Contact* de la figura 6 se presentarán con el formato definido en su clase *Contact*. Además las subclases de una clase *simplexon*, heredarán el formato de presentación definido en la misma. Por otra parte, cuando en la estructura de una instancia de *complexon* se referencie una instancia de *simplexon*, parte de la presentación y el contenido de la primera se obtendrá de la instancia de *simplexon*. Por ejemplo *i1EmployeeHtml* deja parte de su definición a las instancias de *simplexon i1Contact*, *htmlDocumentBegin* y *htmlDocumentEnd*.

Para su implementación el modelo se basa en una extensión del estándar RDF llamada XWMF (*Extensible Web Modeling Framework*).

4.3 Tiles Model

TM defiende la reutilización de partes de páginas *Web* definiendo unidades que llama *tiles* [3]. Un *tile* no es más que una parte de una página Web, la cabecera, el pie de página, un menú, etc. Estos *tiles* son reutilizados y ensamblados usando un patrón, que también es un *tile*. En la figura 7 el contenido de los *tiles header.jsp* y *helloBody.jsp* describen el contenido y la presentación de ciertas partes de una página *Web*, y el *tile myLayout.jsp* describe el patrón con el formato general de la misma. Además en éste, definiendo atributos, se indica el lugar donde se reutilizará el contenido de otros *tiles*. Para concretar cuáles serán exactamente se define un último *tile*, en este ejemplo, *basicPage.jsp*. Aquí se indica el nombre del fichero del patrón y, mediante la etiqueta *tiles:put*, se asocia el nombre de cada atributo definido en *myLayout.jsp* con el nombre del fichero del *tile* correspondiente.

Aunque no podemos extendernos con más ejemplos para mostrarlo (se recomienda acudir a [3]), el modelo también ofrece la posibilidad de adaptar y extender *tiles*. Al reutilizar un *tile* se puede adaptar modificando el valor asignado a alguno de sus atributos. Por otra parte, se pueden definir nuevos *tiles* extendiendo la definición de otro, es decir, utilizándolo como prototipo para luego añadir nuevos atributos o modificar el valor asignado a alguno de los ya existentes.

4.4 Web Service User Interface

WSUI [10] es una propuesta de estándar para el nivel de las interfaces de usuario, es decir, el nivel de presentación, de los servicios *Web*. El objetivo de WSUI es crear un estándar que permita a los servicios *Web* remotos, de tipo XML, SOAP y WSDL, ser integrados en una interfaz de usuario y mostrarlos en

una aplicación *Web*. Un componente WSUI representa una aplicación sencilla que contiene un flujo de tareas y que se presenta por una serie de vistas multipáginas. En la figura 8 se muestra un ejemplo de componente WSUI. Su definición consta de tres partes principales. Con la etiqueta *services* se describe el servicio *Web*, en este caso de tipo SOAP, sobre el que se definirá la interfaz. La etiqueta *views* sirve para indicar cuáles serán las vistas de dicha interfaz, cuáles serán las páginas que se irán presentando durante la ejecución del servicio *Web*. Finalmente con la etiqueta *events* se indica cuáles son los eventos a los que responderá el componente y qué acciones (por ejemplo mostrar alguna de las vistas anteriores o invocar la ejecución del servicio *Web*) están asociadas a los mismos.

```
header.jsp:
    <a href="http://www.mycompany.com">
        <img src="/tutorial/images/headerLeft-logo.gif" align="left" border="0"></a>
    <img src="/tutorial /images/headerRight-logo.gif" align="right" border="0">

helloBody.jsp:
    <div align="center"> <font size="+1"><b>Hello the World</b></font> </div>

myLayout.jsp:
    <%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
    <html>
    <head> ... </head>
    <body>
        <TABLE width="100%">
          <TR><TD colspan="2"><tiles:insert attribute="header" /></TD></TR>
          <TR><TD><tiles:insert attribute="body" /></TD></TR>
              ...
        </TABLE>
    </body>
    </html>

basicPage.jsp:
    <%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
    <tiles:insert page="/basic/myLayout.jsp" flush="true">
       <tiles:put name="header" value="/tutorial/common/header.jsp" />
       <tiles:put name="body"   value="/tutorial/basic/helloBody.jsp" />
        ...
    </tiles:insert>
```

Figura 7. Definición de *Tiles.*

```
<component
   ...
   <title>Stock Quote Lookup</title>
   <description>Look up latest stock prices.</description>
   <services>  <soap-service   id="getQuote"   method="getQuote"   ...  </services>
   <views>
      <view id="start">
            <render mime="text/html" page="MAIN" template="start.xsl"/>
      </view>
      <view id="results">
            <render mime="text/html" page="GENERIC" template="results.xsl"/>
      </view>
      ...
   </views>
   <events>
      <event id="start"> <actions>   <display view="start"/>  </actions>  </event>
      <event id="results">
         <variables>  ...  </variables>
         <actions>
            <invoke service="getQuote" result-variable="QuoteResponse">
               <parameters>  ...  </parameters>
            </invoke>
            <condition expression="$QuoteResponse//return">
               <actions> <display view="results"/>  </actions>
            </condition>
            <display view="error"/>
         </actions>
      </event>
   </events>
</component>
```

Figura 8. Definición de componente WSUI.

En esta propuesta, a diferencia de las anteriores, es el servicio *Web*, con su lógica de negocio, flujo de tareas e interfaz de usuario, lo que se reutiliza. Basta definir una página HTML (ver figura 9) que tenga una instrucción (*includeComponent*) que hace que se genere el evento *start* del componente cuando se cargue la página. Como se ve en la definición del componente WSUI (figura 8) este evento mostrará una vista (descrita en *start.xsl*) y de ahí se seguirá la secuencia de la lógica de negocio.

```
<%@ page import="org.wsui.*, java.util.*" %>
   ...
<html><head>  ...  </head>
<body bgcolor="#ffffff" text="#000000" link="#ff0000" vlink="#800000" ... >
   ...
<%  ComponentManager.getInstance().includeComponent(component_id, request,...);  %>
   ...
</body></html>
```

Figura 9. Reutilización de un componente WSUI.

4.5 Comparación entre modelos

Tras estudiar las características de los modelos anteriores, podemos distinguir cuatro aspectos básicos que pueden servir para diferenciar y clasificar los diferentes mecanismos de reutilización. A continuación describimos las diferentes opciones que podemos encontrar en cada uno de estos aspectos.

- *Qué se reutiliza*. Describe cuáles son los aspectos que el diseñador de una aplicación *Web* puede reutilizar de otras. Una primera opción es reutilizar la plantilla de la página, la que describe qué partes (cabecera, cuerpo, etc.) tiene ésta, dónde debe colocarse un logotipo, etc. Independientemente de la plantilla o estructura, también es posible reutilizar aspectos del contenido de una página. Éstos son los datos que muestra la página, los formatos de presentación y navegación de los mismos y la definición del tipo de seguimiento o traza realizada sobre los datos. En algunos modelos (por ejemplo en AtariX, y en menor medida en *WebComposition*) está clara la diferencia entre estos aspectos y la posibilidad de reutilizarlos de manera independiente, mientras que en otros, como el modelo *Tiles*, la reutilización de los datos se hace implícita al reutilizar la presentación (código HTML). Otro de los aspectos reutilizables a tener en cuenta es el del flujo de tareas de un servicio.
- *Ámbito* de la reutilización. Con este aspecto se indica cuál es el entorno en el que se muestra la reutilización, es decir, dónde ve el usuario la parte reutilizada. Se distinguen dos alternativas: la página *Web* o un conjunto de páginas *Web*. Cuando al definir la reutilización de un elemento (datos, presentación, etc.) varias páginas lo reflejan simultáneamente, decimos que el ámbito de la reutilización es el conjunto de páginas *Web*. Cuando la reutilización se refleja sólo en una página *Web*, y para que se vea en otra más, el constructor de la reutilización correspondiente debe definirse también para ésta, entonces el ámbito es la página *Web*.
- *Cómo se reutiliza*. Este aspecto indica cuál es el mecanismo que utiliza el modelo para implementar el proceso de reutilización. Con el mecanismo de la herencia se permite la definición de clases y subclases, y éstas heredan, reutilizan, las definiciones de sus superclases. Además al crear instancias de una clase éstas reutilizan las definiciones de la misma. En el mecanismo basado en prototipos no existe la noción de clase, todos son objetos. Cuando en la definición de un objeto se utiliza otro como prototipo, las características de éste son copiadas para crear el primero. Los dos objetos serán independientes uno del otro. A diferencia de los anteriores, el mecanismo por delegación permitirá que dos objetos independientes colaboren, de forma que durante la ejecución uno de ellos pueda delegar una parte de la presentación al otro.
- *Adaptación*. Este aspecto estudia si el mecanismo de reutilización permite al diseñador realizar una adaptación previa de la unidad reutilizada.

Con el objetivo de comparar nuestro modelo de reutilización para las UDs con las propuestas que hemos presentado en los apartados anteriores, resumimos en la tabla de la figura 10 las características de todos los modelos en relación con los parámetros anteriores.

5 Conclusiones

El desarrollo de aplicaciones *Web* precisa la adopción urgente de principios establecidos en ingeniería de software. Entre estos, las técnicas de reutilización cuentan entre sus ventajas con la reducción de tiempo

de producción. Tales ventajas pueden ser de utilidad para afrontar los ciclos de vida más cortos y la evolución de los requisitos que caracterizan a las aplicaciones *Web* [8].

Este trabajo introduce el modelo de UD, que tiene en cuenta los requisitos de la *Web*. Una UD es un documento que contiene datos del dominio y puede ser instalada y desarrollada como una unidad significativa. Las unidades de datos se describen utilizando el lenguaje de marcas AtariX.

Proponemos un mecanismo de reutilización (IMPORT) basado en prototipos, cuya unidad de información son los elementos de las unidades de datos. El desarrollador tiene la posibilidad de sustituir cualquiera de las tres dimensiones asociadas a una UD reutilizada, y de emplear el formato de páginas de la aplicación receptora. También proponemos un mecanismo de adaptación que permite al desarrollador de una UD limitar, definiendo una serie de opciones, las posibilidades de adaptarla.

Somos conscientes de las limitaciones de esta propuesta, ya que se centra en aplicaciones *Web* basadas en el contenido. Los siguientes pasos en nuestro trabajo tratarán de incorporar servicios que incluyan lógica de negocio.

Nuestra propuesta se basa en XML, que ha sido utilizado no sólo como notación, sino como nuestro modelo de datos. La implementación de las dimensiones de presentación y navegación se basan en los estándares XSL y XPath. El modelo de ejecución (AtariX) también se basa en XML. El objetivo final es promover el uso de esta propuesta para aplicaciones *Web* basadas en el contenido.

Reutilización	**WCCM**	**WOCM**	**TM**	**WSUI**	**AtariX**
Qué se reutiliza	Datos Presentación	Datos Presentación	Presentación Plantilla	Presentación Flujo de tareas	Datos Presentación Navegación Traza
Ámbito	Página *Web*	Página *Web*	Página *Web*	Conjunto de páginas *Web*	Conjunto de páginas *Web*
Cómo se reutiliza	Prototipo	Herencia	Prototipo Delegación	Delegación	Prototipo
Adaptación	Sí	No	Sí	No	Sí

Figura 10. Comparación de los modelos de reutilización

Bibliografía

[1] S. Ceri, P. Fraternalli, A. Bongio. Web modeling language (webml): a modeling language for designing web sites. *Computer Networks*, 33(1-6):137-157, 2.000.

[2] O. Díaz, J.J. Rodríguez, F. Ibáñez, M. Larrañaga. A client-intensive, model-based approach to web application development: the atarix experience. *Submmited for publication*, 2.001.

[3] C. Dumoulin. http://www.lifl.fr/~dumoulin/tiles. Septiembre 2.001.

[4] H-W. Gellersen, M. Gaedke. Object-oriented web application development. *IEEE Internet Computing*, páginas 60-68, Enero/Febrero, 1.999.

[5] A. Ginige, S. Murugesan. Web Engineering: An introduction. *IEEE Multimedia*, páginas 14-18, Enero/Marzo, 2.001.

[6] R. Klapsing, G. Neumann, W. Conen. Semantics in web engineering: Applying the resource description framework. *IEEE Multimedia*, páginas 62-68, Abril-Junio, 2.001.

[7] C.W. Krueger. Software Reuse. *ACM Computing Surveys*, 24:131-183, Junio, 1.992.

[8] S.P. Overmayer. What's different about requirements engineering for web sites?. *Requirements Engineering*, 5:62-65, 2.000.

[9] J. Sametinger. Software Engineering with Reusable Components. Springer-Verlag, 1.997.

[10] WSUI. http://www.wsui.org/. Septiembre 2.001.

[11] W3c.XML-Data at http://www.w3.org/TR/1998/NOTE-XML-data-0105, 2.000.

[12] W3c.XML Path Language (XPath) Versión 1.0 at http://www.w3.org/TR/xpath.html, 1.999.