

# Personalización de Páginas Web Dinámicas Mediante Ejemplos

José A. Macías y Pablo Castells

E.T.S. Informática, Universidad Autónoma de Madrid  
Campus de Cantoblanco. 28049 Madrid, Spain  
{j.macias,pablo.castells}@uam.es  
<http://www.ii.uam.es/~jamacias,~castells>

**Resumen.** Proponemos una herramienta, DESK, para la modificación del diseño y contenidos de páginas web dinámicas mediante la edición directa del código HTML generado. La herramienta se basa en un modelo del conocimiento del dominio de la aplicación, junto con un modelo explícito del diseño de página, ambos tomados de PEGASUS, un sistema de generación dinámica de páginas web. DESK utiliza heurísticas de desambiguación basadas en estos modelos para inferir la intención del usuario y generalizar los cambios efectuados por él, haciéndolos repercutir en el propio procedimiento de generación de páginas. El mecanismo de inferencia se basa en un modelo estructurado de diferencias que pone en relación los cambios encontrados en los documentos HTML con la semántica del dominio. La desambiguación se ayuda así mismo de un modelo de rol del usuario para decidir en qué parte de la aplicación se deben aplicar los cambios.

## 1 Introducción

Cada día una mayor proporción de páginas en portales y servicios web son generadas dinámicamente. La proliferación de diferentes lenguajes y estándares para el desarrollo de páginas dinámicas (CGI's, Servlets, ASP, JSP, XSLT, Applets, JavaScript, etc.) y su complejidad de uso suponen un problema para el autor medio (expertos del dominio, especialistas en multimedia, etc.) no especializado en el uso de lenguajes de programación. El uso de estas tecnologías exige una gran inversión en aprendizaje o la dependencia de programadores y personal técnico.

Desde principios de los 90 se ha avanzado considerablemente en el desarrollo de sistemas y herramientas que simplifican la generación de páginas dinámicas, a cambio de limitar parcialmente la expresividad proporcionada al desarrollador (ver [1, 2, 14, 15], por citar algunos). Con ellas el autor introduce el conocimiento y los recursos multimedia necesarios para la aplicación en un formato muy sencillo, y el sistema se ocupa de la generación dinámica del código HTML. Por lo general estos sistemas generan código con arreglo a un diseño de página predeterminado, que el diseñador no puede modificar. Para cambiar el diseño sería necesario editar el propio código de la herramienta, o en el mejor de los casos, utilizar un lenguaje específico de modelado del diseño [4, 5].

En este artículo describimos DESK (Dynamic web documents by Example using Semantic Knowledge), una herramienta que permite llevar a cabo un conjunto significativo de modificaciones tanto de los contenidos como del diseño de páginas dinámicas, utilizando un editor estándar de HTML. Con esta herramienta, el usuario parte de una página concreta generada por el sistema y la modifica, proporcionando un ejemplo del cambio que desea. A partir de este ejemplo y de la página original, DESK modifica el procedimiento según el cual se generan las páginas para ajustarlo a lo que la herramienta interpreta que desea el usuario.

DESK está basado en el sistema PEGASUS [4, 5], una herramienta para la generación dinámica de documentos web orientada al desarrollo de aplicaciones educativas y sistemas de información (bibliotecas digitales, museos virtuales, información turística, etc.), que soporta aspectos dinámicos tales como la adaptatividad al usuario y otros tipos de dependencia entre la presentación de contenidos y el contexto en que se ejecuta la aplicación. PEGASUS se basa en una representación estructurada del conocimiento, junto con un modelo explícito de diferentes diseños de página por tipos de contenido, a partir de lo cual PEGASUS genera dinámicamente los documentos web.

El modelado explícito en PEGASUS tanto del conocimiento utilizado en la generación de páginas como de la forma en que el conocimiento se debe presentar al usuario, y la separación entre contenidos y presentación, hacen posible que una herramienta como DESK acceda a esta información y razone sobre ella para buscar un sentido a las modificaciones realizadas por el usuario sobre las páginas generadas. DESK localiza los elementos del dominio y la presentación que han dado origen al código HTML que el usuario modifica, y utiliza estos dos modelos (dominio y presentación) para deducir cambios en los contenidos o en el diseño según las diferencias observadas entre el documento modificado y el original. DESK se ayuda así mismo de información sobre el propio usuario para interpretar correctamente el alcance que se debe dar a los cambios.

## 2 PEGASUS

La autoría de aplicaciones con PEGASUS se basa en una representación explícita del conocimiento del dominio en forma de red semántica de unidades de información con estructura libre (es decir, definida por el autor), algunas de las cuales apuntan a fragmentos de información literal en HTML. El tipo, la estructura y las relaciones de las unidades de conocimiento se describen en una ontología explícita, que consiste en una jerarquía de clases con atributos y relaciones, y que el autor tiene la libertad de variar o definir desde cero. PEGASUS utiliza además un modelo explícito del diseño de las páginas a generar, independiente de los contenidos de las mismas, mediante el cual un diseñador puede modificar el aspecto y la estructura visual de las páginas que se generan. PEGASUS permite utilizar así mismo un modelo del usuario y establecer dependencias en los modelos de dominio y presentación con respecto a las características del usuario.

Una vez definidos la ontología, el modelo del dominio, el modelo de la presentación y el modelo del usuario, el sistema runtime de PEGASUS recibe peticiones HTTP del usuario que indican el acceso a unidades de conocimiento del dominio, y el sistema responde componiendo páginas web para las unidades referidas, utilizando el modelo de la presentación (ver figura 1).

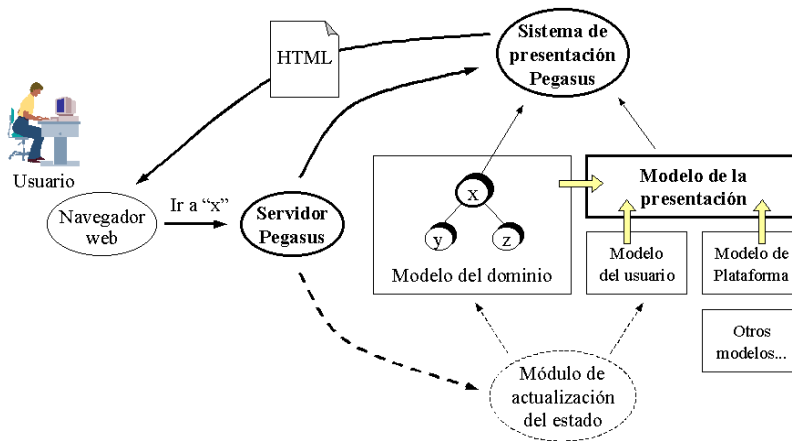


Figura 1 La arquitectura de PEGASUS

### 2.1 Modelo del dominio

El modelo del dominio se construye creando instancias de clases de la ontología y estableciendo relaciones entre ellas. PEGASUS incluye una herramienta de autor [11] para la construcción interactiva de este modelo, y su almacenamiento en documentos XML. Por ejemplo, suponiendo que se han definido clases como *Painter*, *Artwork*, y *ArtisticStyle* para un catálogo de una exposición virtual, el ejemplo siguiente muestra una versión simplificada de una unidad de clase *Painter* que representa conocimiento sobre van Gogh.

```
<Painter id="vangogh" name="Vincent van Gogh"
  birth="1853" death="1890" nationality="Dutch">
  <school> <ArtisticStyle ref="postimpressionism"/> </school>
  <picture> <AtomicFragment url="vangogh-picture.jpg"/> </picture>
  <shortIntro>
    <AtomicFragment> Generally considered the greatest Dutch painter after
    Rembrandt, he powerfully influenced the current of Expressionism in modern
    art. His work is characterized by the striking colour, coarse brushwork,
    and contoured forms. Among his masterpieces are numerous self-portraits
    and the well-known The Starry Night (1889). </AtomicFragment>
  </shortIntro>
  <biography> <AtomicFragment url="vangogh-bio.html"/> </biography>
  <works>
    <Artwork ref="starrynight"/>
    <Artwork ref="sunflowers1"/>
    <Artwork ref="irises"/>
  </works>
```

</Painter>

Los atributos XML como `name` y `birth` corresponden a propiedades de la unidad de conocimiento, mientras que `school`, `picture`, `shortIntro`, `biography` y `works` son relaciones con otras unidades (el atributo `ref` corresponde al identificador de las mismas). Los fragmentos literales se pueden insertar como elementos XML (como `shortIntro`), o almacenar en ficheros externos referenciados mediante su URL en el código XML (como `picture` y `biography`).

## 2.2 Modelo de la presentación

El modelo de la presentación de PEGASUS consiste esencialmente en plantillas de diseño asociadas a las clases de la ontología. Las plantillas determinan qué partes (atributos y relaciones) de una unidad de conocimiento deben ser incluidas en su presentación y en qué orden, su apariencia visual, y la estructura de página. Las plantillas se definen mediante una extensión de HTML basada en JavaServer Pages (JSP). En ellas el diseñador puede hacer uso de todos los elementos de presentación del lenguaje HTML (listas, tablas, frames, enlaces, formularios, etc.), insertando en el mismo, mediante expresiones Java muy sencillas (entre `<%=` y `%>`), los elementos del dominio a presentar. Por ejemplo, una plantilla sencilla para la clase *Painter* podría ser la siguiente:

```
<center>
<h2> <%= name %> </h2>
(<%= nationality %>, <%= birth %> - <%= death %>) <br>
</center><br>
<center><table>
<tr><td valign="top" rowspan="5"> <%= picture %> </td>
<td valign="top"> <%= shortIntro %> </td></tr>
<tr><td> <%= biography %> </td></tr>
<tr><td> <%= works %> </td></tr>
<tr><td> <%= school %> </td></tr>
</table></center>
```

En estas plantillas el autor de la presentación sólo tiene que referenciar atributos y relaciones de la clase presentada (en el ejemplo, en negrita), y PEGASUS se ocupa internamente de acceder a los valores y unidades referenciadas, y elaborar recursivamente su presentación. La página web generada para la unidad sobre van Gogh con esta plantilla de presentación se puede ver en la figura 2.

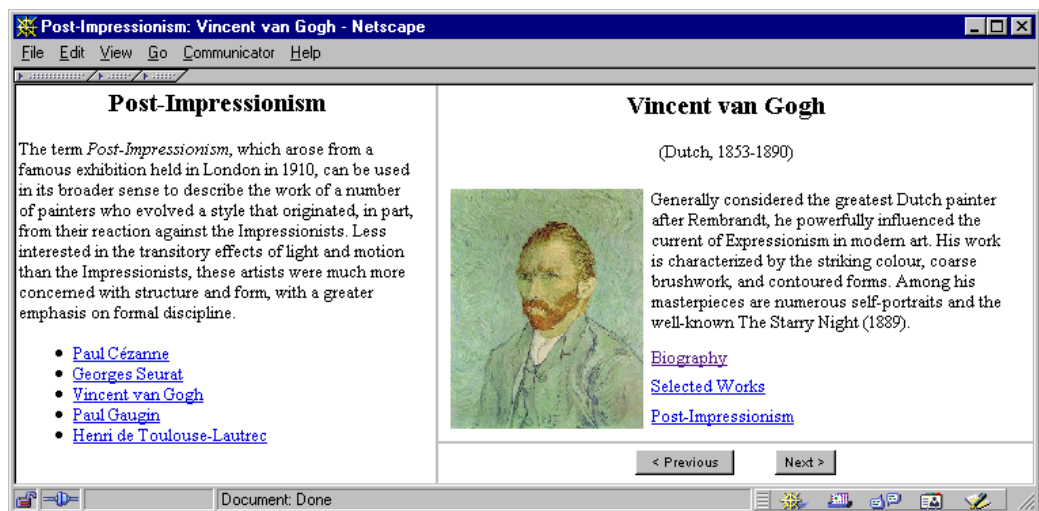


Figura 2 Página generada para una unidad de conocimiento de tipo *Painter*

Partiendo de PEGASUS, el objetivo de DESK es permitir que un diseñador pueda cambiar el diseño de una página como ésta sin tener que editar directamente la plantilla de la presentación mostrada más arriba, y de tal forma que en adelante se utilice el nuevo diseño cuando se generen otras páginas para unidades de este mismo tipo (en este caso, *Painter*).

### 3 DESK: edición mediante ejemplos

DESK permite que cualquier usuario modifique tanto el conocimiento del dominio como el diseño de página editando el código HTML de las páginas generadas por PEGASUS, sin necesidad de utilizar el lenguaje de modelado específico de PEGASUS. A partir de los cambios realizados por el usuario sobre los documentos generados, DESK realiza el camino inverso al recorrido por PEGASUS, retrocediendo hasta los modelos de dominio y presentación, y localizando en PEGASUS los elementos que han dado origen al código HTML que el usuario modifica. Utilizando diferentes heurísticas y técnicas de programación mediante ejemplos para razonar sobre estos dos modelos (dominio y presentación), DESK deduce cambios en los contenidos o en el diseño según las diferencias observadas entre el documento modificado y el original.

PEGASUS implementa una transformación sobre un dominio  $D$  y un modelo de presentación  $P$ , cuyo resultado es la generación de una página web. Para una unidad de conocimiento  $x \in D$ , PEGASUS genera un documento  $D_{HTML}$ . Si representamos esta transformación como una función  $f(D, P, x) \rightarrow D_{HTML}$ , DESK representa una función de sentido inverso  $f^{-1}_{D, P, x}(D_{HTML}, D'_{HTML}) \rightarrow D', P'$ , donde  $D'_{HTML}$  es el documento modificado por el usuario, y  $D'$  y  $P'$  son los modelos de dominio y presentación una vez modificados por DESK.

En general este camino inverso presenta un nivel, mayor o menor, de ambigüedad, en cuanto a que para una misma modificación del usuario pueden haber diferentes interpretaciones y distintas formas de generalización. DESK utiliza heurísticas de desambiguación y aprovecha el conocimiento disponible en PEGASUS, solicitando ayuda del usuario en algunos casos, pero no puede garantizar el acierto en todos los casos. De hecho no siempre existe una solución. Por ejemplo en un caso extremo,  $D'_{HTML}$  podría ser un documento vacío, o completamente diferente al original, en cuyo caso DESK no podría deducir nada. El objetivo del trabajo propuesto no es desarrollar un sistema infalible, sino alcanzar un grado de acierto suficiente para una herramienta útil.

Los tipos de cambios que DESK es capaz de inferir incluyen:

- Inserción, eliminación y modificación de fragmentos literales de HTML en cualquier punto de una plantilla de presentación.
- Inserción, eliminación y modificación de etiquetas HTML que rodean a referencias a elementos del dominio, dentro de una plantilla de presentación.
- Eliminación de referencias a elementos del dominio dentro de una plantilla de presentación.
- Modificación de fragmentos de texto y multimedia (HTML) dentro del modelo del dominio.

La diferenciación y realización de un tipo u otro de cambio es inferida por el sistema, teniendo en cuenta, entre otros factores, el rol del usuario que manipula la herramienta. Actualmente DESK no permite reconocer otro tipo de cambios más allá de los citados. En particular, DESK no puede modificar la estructura del dominio (relaciones entre las unidades de conocimiento), ni incluir nuevas partes de una unidad en su plantilla de presentación. Este tipo de modificaciones requerirían cambio cualitativo en la interacción con el usuario, y un editor de HTML aumentado con nuevas funcionalidades con las que el usuario pudiera, por ejemplo, visualizar directamente el modelo del dominio.

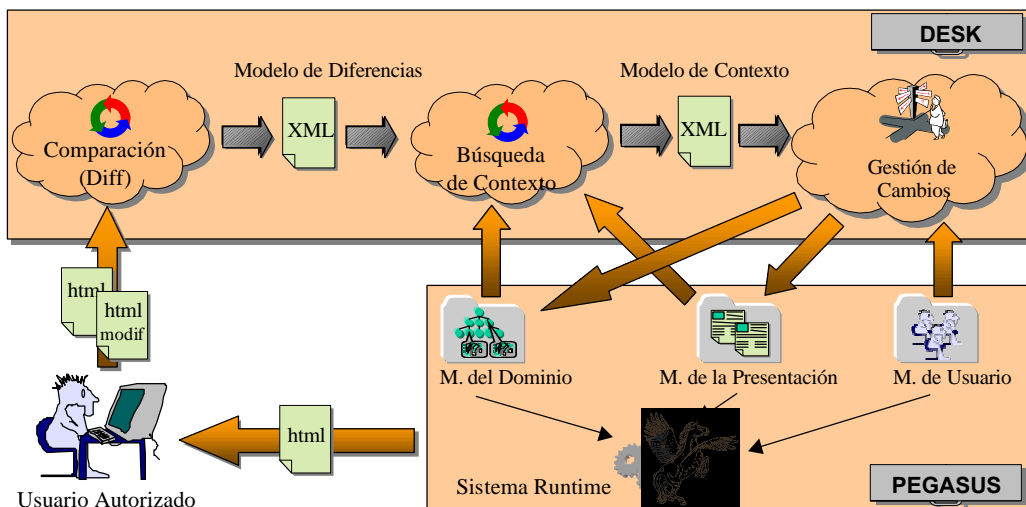


Figura 3 Esquema de la herramienta DESK puesta en contexto

En la figura 3 podemos ver la arquitectura de DESK. En primer lugar, un diseñador o usuario autorizado modifica la página generada por PEGASUS, y envía a DESK tanto la página HTML generada, como la modificada por él. A partir de estas dos páginas, el módulo de diferencias (“Diff”) de DESK infiere un modelo de diferencias estructurado. El siguiente módulo se encarga, tomando este modelo junto con el del dominio, de dotar de semántica a las diferencias encontradas, elaborando un modelo de contexto. El modelo de contexto permitirá localizar, en el modelo del dominio, información sobre cada una de las modificaciones detectadas en la página, así como aquellos cambios que carezcan de contexto en la presentación (modificación de fragmentos HTML en las plantillas de presentación y/o en unidades de conocimiento). Con esta información, un tercer módulo de gestión de cambios será el encargado de tomar el modelo generado anteriormente y efectuar los cambios en el modelo de la presentación o en el modelo del dominio. A partir de este momento, la próxima vez que se generen páginas se aplicará la nueva presentación. En la sección que sigue mostramos un ejemplo para ilustrar el mecanismo descrito.

## 4 Los modelos DESK

Como ya se comentó anteriormente, DESK utiliza una serie de modelos, codificados en XML, donde se refleja, para cada uno de los pasos descritos con anterioridad, la distinta información a ser utilizada por la herramienta. A parte del modelo del dominio utilizado por PEGASUS, DESK crea y gestiona un modelo de diferencias y otro de contexto.

### 4.1 Modelo de diferencias

El modelo de diferencias permite plasmar, de una forma estructurada, las diferencias sintácticas encontradas durante el proceso de comparación de los ficheros enviados por el diseñador.

```
<DifferenceModel>
  <Diff id="1">
    <Original line="38" action="change">
      <h2> Vicent van Gogh </h2>
    </Original>
    <Modified line="38" action="change">
      <h1> <u> Vicent van Gogh </u> </h1>
    </Modified>
  </Diff>
  <Diff id="2">
    <Original line="38" action="change">
      <h2> Dutch </h2>
    </Original>
    <Modified line="38" action="change">
      <h2> Holandés </h2>
    </Modified>
  </Diff>
  <Diff id="3">
    <Modified line="60" action="adition">
      <blockquote> <h2> Related Schools </h2> </blockquote>
    </Modified>
  </Diff>
  .....
</DifferenceModel>
```

El código anterior muestra un fragmento de la especificación XML utilizada para codificar el modelo de diferencias. Para cada una de las diferencias apreciadas en el fichero modificado, se generará un elemento “Diff”, donde se verá reflejado el estado del texto antes de ser modificado (“Original”) y después de ser modificado (“Modified”). Cada elemento contiene un atributo “action” que refleja el tipo de cambio producido en dicho fragmento (añadir, eliminar, cambiar). En este ejemplo DESK ha encontrado, en primer lugar, que el usuario ha modificado la apariencia del literal “Vicent van Gogh” (<Diff id="1">), añadiendo un subrayado <u> y aumentando el nivel del título a <h1>. La segunda modificación (<Diff id="2">) indica que el usuario ha cambiado “Dutch” por “Holandés”, y la tercera (<Diff id="3">) refleja que el usuario ha añadido una nueva línea de código HTML.

## 4.2 Modelo de contexto

El modelo de contexto aporta información semántica asociada a cada una de las modificaciones encontradas en el modelo de diferencias. De esta forma se podrán localizar exactamente, en el modelo del dominio y de la presentación, los cambios a efectuar, incluyendo cambios en etiquetas HTML aplicadas a elementos del dominio en el modelo de la presentación, así como la eliminación de estos elementos.

```

<ContextModel>
  <Diff_context id="1">
    <Context class_name="Painter" attribute_name="name" />
    <Tags text_found="Vicent van Gogh"> <h1/> <u/> </Tags>
  </Diff_context>
  <Diff_context id="2">
    <Context class_name="Painter" attribute_name="nationality" />
    <ReplaceBy> Holandés </ReplaceBy>
  </Diff_context>
  <Diff_context id="3">
    <Location place="after"> <%= school %> </Location>
    <Insert> <blockquote> <h2> Related Schools </h2> </blockquote> </Insert>
  </Diff_context>
  .....
</ContextModel>

```

En el código anterior se muestra un fragmento en XML del modelo de contexto generado. Este modelo se construye utilizando la ontología del dominio para localizar datos en la red de unidades de conocimiento, como la clase y el nombre del atributo que contiene la cadena, etc. En este caso, el contexto de la modificación sobre la apariencia del texto “Vicent van Gogh” queda reflejado en el “Diff\_context” número 1, encontrado dentro del atributo “name” de la clase “Painter” de la ontología. Dentro del elemento “Tags” se recogen las nuevas etiquetas HTML añadidas a la presentación, según el modelo de diferencias. En la segunda entrada del contexto (<Diff\_context id="2">) tenemos modelizado un cambio sobre el modelo del dominio, localizando igualmente la clase y el atributo donde se ha encontrado el literal. En el tercer caso (<Diff\_context id="3">), el contexto hace referencia directamente a un cambio sobre el modelo de la presentación, donde debe llevarse a cabo la inserción del nuevo código HTML, justo después del elemento “school” en la plantilla de presentación (<Location place="after">). Esta información es necesaria debido a que no existe una relación directa entre la página modificada y el modelo de la presentación, siendo la herramienta la encargada de inferir correspondencias entre la distinta ubicación de las inserciones, borrados y actualizaciones realizadas por el usuario en la página HTML, y por la herramienta en el modelo de la presentación.

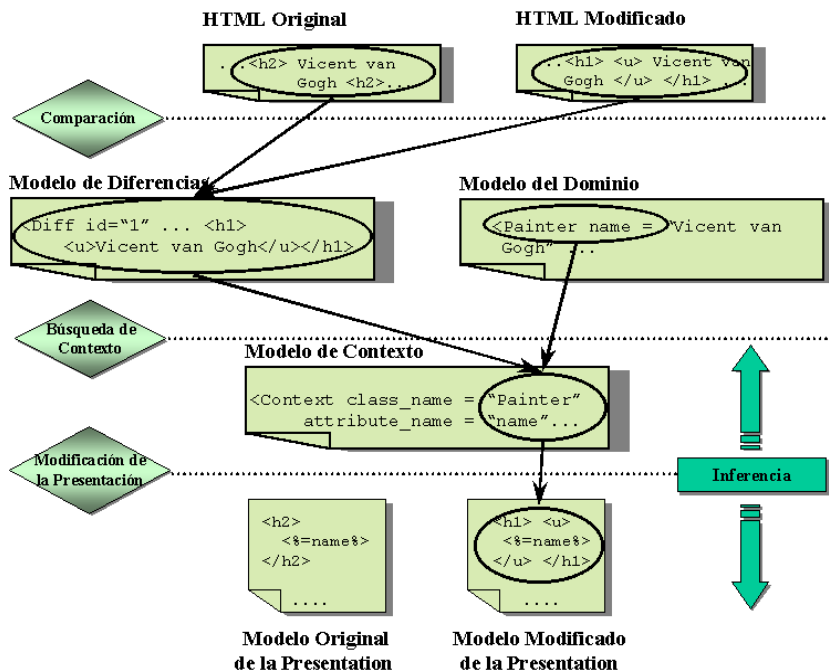


Figura 4 Diagrama detallado del funcionamiento de DESK

Después de construir cada uno de estos modelos, un último módulo será el encargado de aplicar el modelo de contexto para acometer cada una de las modificaciones tanto en la apariencia de la presentación como en los contenidos.

En la figura 4 se detalla, a un nivel de abstracción mucho menor que el utilizado anteriormente, un resumen de cada uno de los pasos seguidos en la obtención de cada modelo. De esta forma podemos ver cómo para cambiar el aspecto de la línea "Vicent van Gogh" obtenemos, en primer lugar, qué cosas se han modificado realmente a partir de los ficheros HTML suministrados por el usuario. Posteriormente se trata de encontrar el contexto de lo modificado a partir del modelo del dominio, y finalmente se modifica la presentación teniendo en cuenta todo lo anterior. También podemos observar cómo los mecanismos de inferencia son aplicados tanto a nivel de contexto como en la propia modificación de la presentación.

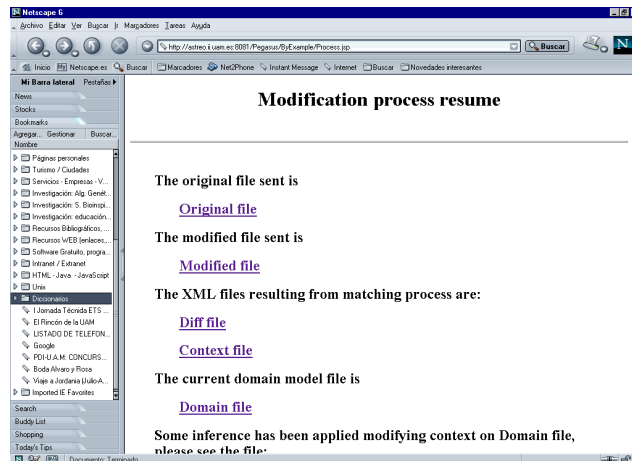


Figura 5 Página de resultados de DESK

Una vez que los cambios sobre la presentación han sido realizados, al diseñador se le reporta una página como la que aparece en la figura 5, donde se ofrece información sobre los cambios realizados, los distintos modelos comentados anteriormente, un informe de eventos ocurridos durante el proceso, posible errores, así como información sobre el mecanismo de inferencia, informando sobre las decisiones tomadas por la herramienta, y por último, pidiendo la confirmación del usuario para llevar a cabo los cambios sobre el modelo interno. Esta página se utiliza así mismo en algunos casos para permitir la intervención del usuario en decisiones delicadas en las que DESK encuentra difícil aplicar un criterio de elección.

### 4.3 Roles de usuario

DESK incluye en el modelo de usuario atributos que definen distintos tipos de roles. Hemos considerado tres roles básicos de operación bajo DESK: editor, capacitado para modificar contenidos; diseñador, autorizado para modificar la presentación; y usuario del sistema o usuario final, al cual van destinadas las páginas generadas. DESK tiene en cuenta en todo momento el perfil del usuario para efectuar cada una de las operaciones descritas anteriormente. Esta información se recoge en un modelo como el que se muestra a continuación, reflejando el tipo de usuario que está interactuando con el sistema.

```
<User name="Jose A. Macias" age="28" language="Spanish">
  <login> macias </login>
  <role> designer </role>
  <profile> editor </profile>
  .....
</User>
```

Dependiendo del rol, el nivel de operación del usuario puede variar. El sistema detecta esta característica y avisa al usuario de su nivel de permisos sobre la aplicación, permitiéndole, en su caso, modificar la página con su navegador y enviar a DESK la versión original y modificada de la página en cuestión. Si además el usuario cuenta con un perfil de "editor", el mecanismo de inferencia de DESK permitirá realizar cambios, no solamente en el modelo de la presentación (apariencia), sino también en el modelo del dominio (contenido), tal y como se explicó en apartados anteriores. Este modelo de usuario es utilizado, de la misma forma, por PEGASUS para

adaptar la presentación a un usuario en concreto a partir de una identificación previa de la persona que manipula la presentación.

## 5 Trabajo relacionado

El desarrollo de herramientas de autor WYSIWYG para páginas dinámicas es un problema inherentemente difícil. Herramientas como las de desarrollo de sistemas hipermedia adaptativos [1, 2, 14, 15] incluyen editores interactivos para introducir contenidos y definir estructuras de conocimiento complejas [10], pero no para el diseño de la presentación y su relación con otros modelos (dominio, usuario, plataforma, etc.).

En DESK hemos abordado este problema con el enfoque de las llamadas técnicas de programación mediante ejemplos [5, 6, 7], que consisten en inferir información procedural a partir de ejemplos de lo que se desea producir. La programación mediante ejemplos conlleva una ambigüedad inherente al hecho de que se está derivando información general a partir de casos particulares proporcionados por un humano. Para resolver esta ambigüedad se han utilizado en otros sistemas estrategias como el seguimiento de las acciones del usuario paso a paso (vs. observar sólo el estado inicial y final), la utilización de varios ejemplos, ejemplos negativos, o la petición interactiva de ayuda al usuario, entre otras.

DESK razona a partir de un solo ejemplo de estados inicial y final, lo que tiene la ventaja de que el usuario no necesita utilizar un editor específico que monitorice todas sus acciones. A cambio, DESK se ayuda del modelo del dominio para dotar de significado a los cambios detectados. En general, es imposible desarrollar un sistema de estas características que acierte *siempre*, a menos que se impongan tantas restricciones que el sistema pierda toda su generalidad, o se hagan tantas preguntas al usuario que deje de ser fácil de utilizar, por lo que la meta de esta filosofía consiste generalmente en alcanzar un grado de acierto suficiente para que el sistema basado en ejemplos resulte útil.

La extracción heurística de información estructurada (modelo de diferencias y contexto) a partir de un documento semi-estructurado (código HTML) en DESK es similar a la que llevan a cabo los denominados *wrappers* [8, 9, 12], que se vienen utilizando para proporcionar un acceso uniforme a datos que residen en repositorios heterogéneos de información (ficheros, bases de datos, etc.).

## 6 Conclusiones

La herramienta propuesta gestiona automáticamente los cambios producidos por usuarios dentro de una presentación en el entorno de la web. Nuestra herramienta se basa en el sistema PEGASUS, que permite representar contenidos y apariencia por separado. DESK induce los cambios a efectuar en uno u otro modelo teniendo en cuenta el perfil del usuario que interactúa bajo el sistema. La herramienta se ocupa de trasladar los cambios a los modelos internos de forma automática, lo que permite que los conocimientos del diseñador se limiten exclusivamente a los del dominio de la aplicación o al del diseño de página, puesto que la modificación de las páginas puede realizarse con cualquier editor estándar, como por ejemplo la herramienta Composer de Netscape®. La utilización de modelos estructurados para la representación de los cambios permite, de una forma modular, obtener una rica semántica sobre los pasos de actualización e inferencia, así como reutilizar esta información en el futuro para poder desechar cambios efectuados o, simplemente, generar un histórico de actualizaciones a lo largo de la vida útil de la presentación.

DESK y PEGASUS han sido implementados en Java (JDK 1.3), con DOM y JavaServer Pages. Actualmente estamos ampliando DESK para permitir la inferencia de cambios sobre los elementos dinámicos de la presentación que se originan directamente mediante código Java inserto en las plantillas de la presentación. El objetivo es extender la variedad de cambios que se puedan reconocer sobre la presentación, permitiendo inferir cambios más profundos en la parte que afecta directamente a las rutinas encargadas de gestionar las presentaciones. También estamos estudiando otras posibilidades, como la inserción de nuevas unidades de conocimiento por parte del usuario, que modificarían directamente la estructura del modelo del dominio.

El trabajo presentado en este artículo está parcialmente financiado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), proyecto nº TIC2001-0685-C02-01.

## Referencias

1. Brusilovsky, P., Eklund, J., Schwarz, E.: Web-based Education for all: a Tool for the Development of Adaptive Courseware. *Computer Networks and ISDN Systems*, 30, 1-7, 1998.
2. Carro, R. M., Pulido, E., Rodríguez, P.: Dynamic generation of adaptive Internet-based courses. *Journal of Network and Computer Applications* 22, 1999, 249-257.
3. Castells, P., Szekely, P.: Presentation Models by Example. En: Duke, D.J., Puerta A. (eds.): *Design, Specification and Verification of Interactive Systems '99*. Springer-Verlag, 1999, pp. 100-116.
4. Castells, P., Macías, J.A.: Un sistema de presentación dinámica hipertexto para representaciones personalizadas del conocimiento. *Actas del 2º Congreso de Interacción Persona-Ordenador (Interacción 2001)*. Salamanca, Junio 2001.
5. Castells, P. Macías, J.A.: An Adaptive Hypermedia Presentation Modeling System for Custom Knowledge Representations. *Proceedings of the World Conference on the WWW and Internet (WebNet'2001)*. Orlando (Florida), October 2001.
6. *Communications of the ACM*. The Intuitive Beauty of Computer Human Interaction. Special issue on Programming by Demonstration, 43, 3, March 2000.
7. Cypher A. (ed.). *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.
8. Gruser, Jean-Robert et al.: Wrapper Generation for Web Accessible Data Sources. *Proceedings of COOPIS'98*, 1998.
9. Huang, Anita W.: Aurora: A Conceptual Model for Web-Content Adaptation to Support the Universal Usability of Web-based Services. *Conference on Universal Usability (CUU 2000)*. Arlington VA, USA, 2000.
10. Macías, J.A. and Castells, P. Interactive Design of Adaptive Courses. In *Computers and Education – Towards an Interconnected Society*, M. Ortega and J. Bravo (eds.). Kluwer Academic Publishers, Dordrecht (The Netherlands), 2001.
11. Macías, J.A., Castells, P.: An Authoring Tool for Building Adaptive Learning Guidance Systems on the Web. *Lecture Notes in Computer Science: Active Media Technology – AMT 2001*. Springer-Verlag, Viena (Austria), 2001.
12. Muslea, I.: Extraction Patterns for Information Extraction Tasks: A Survey. *Proceedings of AAAI Workshop on Machine Learning for Information Extraction*. Orlando, Florida, July 1999.
13. Vassileva, J.: Dynamic Course Generation on the WWW. *Actas 8th World Conference on Artificial Intelligence in Education (AIED'97)*. Kobe, Japón, 1997, 498-505.
14. Weber, G. and Specht, M.: User modeling and Adaptive Navigation Support in WWW-based Tutoring Systems. *Proceedings 6th International Conference on User Modeling (UM97)*. Sardinia, Italy, 1997.