Un sistema de presentación dinámica en entornos web para representaciones personalizadas del conocimiento

Pablo Castells, José Antonio Macías

E.T.S. de Informática
Universidad Autónoma de Madrid
Campus de Cantoblanco, 28049 Madrid, Spain
+34-91-3482284, +34-91-3482241
{pablo.castells, jose.a.macias}@uam.es

Resumen

Los sistemas web adaptativos presentan información al usuario en documentos hipermedia generados dinámicamente de acuerdo con un modelo automáticamente actualizado del usuario. Por lo general los sistemas adaptativos existentes imponen un modelo fijo para representar el conocimiento que maneja la aplicación. Por otra parte, utilizan patrones invariables de diseño de página para los documentos generados, que no pueden ser libremente configurados por un diseñador humano. En este artículo proponemos un sistema genérico de presentación para sistemas hipermedia de enseñanza adaptativa altamente independiente de la representación del conocimiento del dominio y del mantenimiento del estado de la aplicación. La generalidad se consigue proporcionando un marco de aplicación para la definición de ontologías que mejor se ajustan a un dominio o un autor concreto. La presentación de las páginas a generar se describe en términos de clases y relaciones de la ontología. Un modelo explícito de la presentación, separado de los contenidos del curso, se utiliza para permitir a los diseñadores un extenso control sobre la generación de todos los aspectos de la presentación, con un coste de desarrollo moderado.

Palabras clave: hipermedia adaptativo, ontologías, representación del conocimiento, modelado de usuario, herramientas de diseño de interfaces, enseñanza en la web.

1. Introducción

El rápido desarrollo de los sistemas de formación a través de la web está proporcionando a los estudiantes una autonomía cada vez mayor, al tiempo que las aplicaciones educativas llegan a un público cada vez más amplio y heterogéneo. Este panorama suscita un renovado interés por el desarrollo de sistemas hipermedia con capacidad de adaptación automática a distintos tipos de usuario, plataforma y situación, que tenga en cuenta igualmente la evolución de cada usuario a lo largo del tiempo [Brusilovsky98b]. Una preocupación prioritaria presente explícita o implícitamente en todo el trabajo de investigación llevado a cabo en el campo de la enseñanza adaptativa es la de encontrar una representación apropiada del conocimiento pedagógico [Murray98]. Cada herramienta de construcción de sistemas tutores establece su propia forma de estructurar el dominio, de manera que el diseñador describe la materia en los términos que establece la herramienta, y ésta se hace cargo de la selección, presentación y secuenciación dinámica del material didáctico, así como la interacción con el usuario. En las herramientas existentes los documentos hipermedia se generan según diseños de página prefijados que los autores del curso no pueden configurar (ver [Brusilovsky98a, Carro99], por ejemplo), o bien se

requiere de éstos una descripción de todas las páginas a generar una por una (como en [DeBra98]). El propósito de nuestro trabajo es la creación de un sistema genérico de presentación hipermedia que proporcione a los autores de cursos un paradigma de especificación sencillo para definir elementos no triviales de diseño de página y presentación adaptativa, independientemente de los contenidos [Castells01, Macías01a, Macías01b]. La herramienta que hemos desarrollado, PEGASUS (Presentation modeling Environment for Generic Adaptive hypermedia SUpport Systems), hace mínimas suposiciones sobre cómo se representa el conocimiento educativo, con el objetivo de poder ser utilizada con diferentes formas de representación del dominio, y por lo tanto con diferentes sistemas de construcción y gestión de cursos. Para admitir distintos enfoques, PEGASUS soporta la definición de ontologías del dominio a la medida, para la descripción y estructuración conceptual de la materia (como en [Murray98]). Una vez que se ha definido una ontología, el diseñador construye cursos creando objetos del dominio y relacionándolos entre sí utilizando el vocabulario conceptual definido por la ontología. La composición de las páginas a generar se define mediante un modelo explícito de la presentación donde se asocian presentaciones a clases y relaciones de la ontología. La utilización de un modelo abstracto de la presentación, separado de los contenidos, permite configurar la presentación adaptativa de contenidos independientemente de la elaboración de los mismos.

2. Antecedentes

La representación y utilización explícita de conocimiento semántico sobre un dominio para facilitar o guiar el acceso a la información ha sido una preocupación primordial en el campo de los sistemas hipermedia desde sus inicios [Trigg86]. En la literatura sobre sistemas hipermedia adaptativos se encuentran diversas formas de representar el conocimiento. La mayoría se basan en un nivel de contenidos, discretizado en base a algún tipo de unidad elemental, y un nivel de estructura semántica, que se utiliza como mapa de carreteras para guiar la navegación. Sin embargo, existe una gran variabilidad tanto en la forma de estructurar los contenidos, como en la organización de la red conceptual, y la conexión entre ambos planos.

Por citar unos cuantos, ELM-ART [Weber97] e Interbook [Brusilovsky98a] estructuran los cursos en unidades temáticas: *capítulos*, *secciones*, *subsecciones* y *páginas terminales*, acompañadas por un conjunto de *conceptos* interrelacionados mediante dos tipos de relación: *prerequisito* y *producto* (*outcome*). La relativa sencillez de este modelo de dos relaciones contrasta con la riqueza léxica de herramientas como HyperTutor [Pérez95], donde el mapa conceptual incorpora una amplia variedad de relaciones extraídas de la literatura sobre teoría educativa: *prerequisito*, *secuencia*, *agregación*, *similitud*, *oposición*, *ejemplo*, *caso particular*, y *excepción*. AHA [DeBra98] permite una composición de páginas más flexible, a base de *fragmentos* HTML de inclusión condicional. Los conceptos tienen tres atributos booleanos para indicar el estado del conocimiento del alumno respecto de cada concepto: *conocido*, *leído*, y *listo para ser leído*. AHA contempla cuatro tipos de relación entre conceptos: *prerequisito*, *inhibidor*, *parte-de* y *enlace*. Las relaciones entre conceptos tienen un parámetro que representa un estado (booleano) o una medida cuantitativa. AHM [daSilva98] utiliza un modelo de conceptos y documentos muy similar a AHA, con relaciones de *prerequisito*, *especialización*, y *afinidad*, entre conceptos, e *ilustración* y *evaluación*, entre concepto y documento.

DCG [Vassileva97] y TANGOW [Carro99] se distinguen por generar la estructura del curso, o partes de ella, en tiempo de ejecución. En DCG se define un primer nivel de conceptos interconectados mediante relaciones de *abstracción*, *agregación*, *causalidad*, *analogía*, y *precedencia temporal*, bajo el cual, para cada concepto, se construye un árbol de *tareas* y *subtareas* de aprendizaje con la secuencia de documentos y acciones a seguir por el alumno para aprender el concepto [Vassileva95]. La relación tarea-subtarea puede ser de distintos tipos: *introducción*, *explicación*, *ejemplo*, *ejercicio* y *evaluación*. La descomposición de tareas y la asociación de fragmentos a tareas atómicas se determinan dinámicamente en tiempo de ejecución por medio de *reglas*. TANGOW, a diferencia de DCG, permite también asociar contenidos a las tareas compuestas. En otros sistemas la generación de relaciones semánticas es aún más dinámica y se realiza por mecanismos de búsqueda automática basada en metadatos asociados a las unidades de información [Wilkinson99]. Esta filosofía es útil cuando el espacio de conocimiento es excesivamente amplio y/o volátil para definir y mantener explícitamente las relaciones necesarias.

Fuera del ámbito hipermedia, Eon [Murray98] adopta un punto de vista más general que los sistemas anteriores, permitiendo que el autor defina sus propias categorías de conocimiento (*tópicos*), y las relaciones entre ellas que considere oportunas. A cada tópico se asignan conjuntos de unidades de contenidos mediante distintas relaciones definidas por el diseñador, tales como *introducción*,

explicación, *evaluación*, *nivel básico*, *nivel avanzado*, *resumen*, etc. La selección efectiva de contenidos se hace en tiempo de ejecución aplicando estrategias pedagógicas predefinidas de selección y ordenación (p.e. escoger un elemento al azar, o presentar todos en secuencia). Eon proporciona además una herramienta gráfica donde el diseñador construye interfaces de usuario parametrizadas, a las que asocia unidades de contenido, de tal forma que los *widgets* de la interfaz (botones, tablas, grafos, cuadros de diálogo, etc.) toman valores de la unidad de conocimiento presentada.

Con respecto a la utilización de modelos de presentación explícitos para la generación dinámica de documentos web, [Saiz98] propone un sistema basado en modelos adaptativos del diseño de página para permitir la visualización dinámica en HTML de información almacenada en bases de datos. La restricción más significativa de este sistema en relación con los objetivos que nos hemos propuesto estriba en su limitada capacidad expresiva para la representación del conocimiento, que en dicho sistema se limita a un modelo de datos relacional.

3. Representación del conocimiento

Las diferentes formas de representar el conocimiento descritas en la sección anterior dan como resultado un modelo del dominio que las herramientas utilizan como estructura de referencia para mantener un modelo actualizado del conocimiento y objetivos del alumno en relación con la materia descrita (modelo *overlay*). Esta información se utiliza en tiempo de ejecución para adaptar al usuario la selección y presentación de contenidos y enlaces. Si bien los sistemas citados incluyen distintas formas de control para el autor sobre las estrategias docentes y de actualización del mapa conceptual, la generación de páginas, con la excepción de Eon [Murray98], tiene lugar siguiendo patrones fijos de presentación programados en la herramienta.

Nuestro sistema permite la generación automática de documentos hipermedia del tipo que soportan los sistemas adaptativos descritos, con pleno control para el autor sobre los aspectos visuales (presentación) de los documentos generados, y sin imponer una representación particular del conocimiento. Para ello, igual que Eon, PEGASUS permite la definición de taxonomías a la medida del dominio y del autor. La terminología así definida se utiliza por un lado para la descripción de la materia por parte del autor, y por otro para la construcción de modelos de presentación asociados a las diferentes categorías del conocimiento (Figura 1).

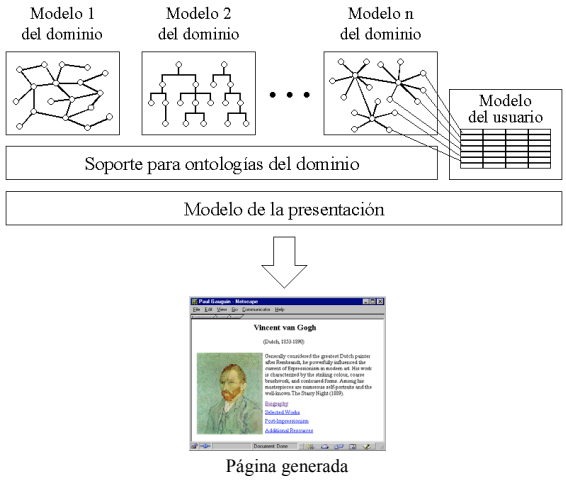

Figura 1. Presentación adaptativa basada en ontologías en PEGASUS

3.1. Ontologías

Una ontología es una representación conceptual compartida que proporciona una comprensión común de un dominio. Las noción de ontología fue desarrollada originalmente en el campo de la Inteligencia Artificial para facilitar la compartición y reutilización del conocimiento [Gruber93]. Más tarde se han utilizado ontologías para la extracción inteligente de conocimiento en la web, como instrumento para modelizar información semántica (metadatos) que se utiliza para anotar documentos web (ver por ejemplo [Fensel99]). En PEGASUS, las ontologías se utilizan para proporcionar la máxima flexibilidad en la representación del conocimiento pedagógico. Son además un elemento esencial para conseguir la separación entre presentación y contenidos.

La ontología del dominio en PEGASUS consiste en un conjunto de clases que mejor se adecuan a un campo de aplicación determinado, o que reflejan la visión particular de un autor sobre el dominio. En nuestro enfoque, las ontologías se pueden definir con un alto grado de libertad, con clases muy genéricas, como Concepto, Lección, o Hecho, o más específicas, como Algoritmo, Teorema, o Definición, según el diseñador lo considere más adecuado. En particular, sería posible definir taxonomías como las de los sistemas descritos en la sección 2.

Las ontologías incluyen elementos para representar información sobre la materia (p.e. un teorema tiene un enunciado y una demostración), información

pedagógica (p.e. las lecciones tienen niveles de dificultad), e información sobre el estado del usuario y del entorno en tiempo ejecución (p.e. un concepto es conocido o no por el estudiante). Todo este conocimiento se recoge mediante la definición de atributos para las clases, y relaciones entre clases.

PEGASUS proporciona una clase raíz, *DomainObject*, y dos subclases predefinidas, *Topic* y *Fragment*, para ser subclasificadas por los diseñadores. Los tópicos se presentan al usuario final en una página separada, mientras que varios fragmentos pueden ser insertados en la misma página. Una clase predefinida de *Fragment*, *AtomicFragment*, almacena contenido multimedia (código HTML). *DomainObject* tiene algunos atributos predefinidos como *name* y *title*, a los que el diseñador puede añadir otros, como *read*, *known* o *visible*, y nuevas relaciones, como *prerequisite* y *subunit*.

Entre otros formatos, PEGASUS permite la representación de las clases de la ontología y las instancias del dominio en XML. El siguiente ejemplo ilustra la definición de una clase Algoritmo con tres relaciones: procedimiento, ejemplos y prueba de corrección. Por brevedad, omitimos aquí otras relaciones que normalmente se incluirían, tales como definiciones previas, motivación, problema a resolver, análisis de complejidad, o aplicaciones.

```
<Class name="Algorithm" parent="Topic">
 <Attribute name="recursive"
            type="Boolean"/>
 <Relation name="procedure"
           type="AtomicFragment"
           multivalued="false"
           title="Procedure"/>
 <Relation name="examples"
           type="AtomicFragment"
           multivalued="true"
           title="Examples"/>
 <Relation name="correction"
           type="Theorem"
           multivalued="true"
           title="Proof of Correction">
    <Attribute name="relevant"
               type="Boolean"/>
    <Attribute name="difficulty"
               type="Number"/>
 </Relation>
</Class>
```

Las relaciones pueden tener atributos propios como, en el ejemplo anterior, la dificultad de la prueba de corrección, que expresan propiedades de la relación en sí. Todas las relaciones tienen un atributo predefinido *title* que se utiliza en algunos casos para generar títulos o texto para enlaces hipermedia.

Además de la ontología del dominio, el diseñador puede definir estructuras de datos más sencillas para describir perfiles de usuario, datos sobre el curso (plan, objetivos, requisitos, duración, nº de alumnos, etc.), características de la plataforma, y otros aspectos considerados relevantes para la adaptatividad del sistema en desarrollo.

3.2. Modelo del dominio

Una vez definida una ontología, los cursos se construyen mediante la creación de redes semánticas de objetos del dominio, utilizando las clases y relaciones definidas en la ontología. El siguiente ejemplo ilustra la creación de una instancia de Algoritmo para representar el algoritmo de Dijkstra para el problema de los caminos de coste mínimo (suponemos que el atributo *title* y la relación *prerequisite* están predefinidos en la clase raíz *DomainObject*).

```
<Algorithm name="Dijkstra"
           title="Dijkstra's Algorithm"
           recursive="false">
  <prerequisites>
     <Algorithm ref="relaxation"/>    (1)
  </prerequisites>
  <procedure>
     <AtomicFragment>                 (2)
       <tt> Dijkstra(G,s) <br>
         Init(G,s) <br>
         Q = V[G] <br>
         while Q not empty do
       <br>   
       u = ExtractMin(Q) <br>
         for v in Adj[u] do
       Relax(G,u,v)</tt>
    </AtomicFragment>
  </procedure>
  <examples>
    <AtomicFragment URL="ex1.html"/> (3)
  </examples>
  <correction relevant="true"
              difficulty="0.6">
     <Theorem ref="theorem1"/>
  </correction>
</Algorithm>
```

Los elementos con el atributo `ref` indican referencias a otras unidades del curso (por ejemplo, la línea 1 se refiere a un algoritmo con `name="relaxation"`). Los fragmentos atómicos pueden consistir directamente en un string, como el procedimiento del algoritmo de Dijkstra (línea 2), o una dirección web, como en la línea 3.

En tiempo de ejecución PEGASUS mantiene para cada usuario una copia de todos los objetos del dominio, donde los atributos de las clases (p.e. *read*) se utilizan para medir el progreso del usuario. Estos valores se pueden utilizar para condicionar la presentación (ver Sección 4), pero su actualización requiere un módulo de actualización complementario (ver arquitectura, Sección 5).

El modelo del dominio en PEGASUS soporta además la definición de elementos adaptativos en el propio modelo mediante la introducción de condiciones sobre cualquier parte de la estructura. Por ejemplo, para seleccionar distintos ejemplos según el nivel del alumno, la línea 3 se cambiaría por:

```
<examples>
  <case condition =
           "user.expertise < 0.5">
    <AtomicFragment URL="ex1.html"/>
  </case>
  <case condition =
           "user.expertise >= 0.5">
    <AtomicFragment URL="ex2.html"/>
  </case>
</examples>
```

De este modo es posible construir estructuras dinámicas como las jerarquías de tareas de Tangow [Carro99] ó DCG [Vassileva97], que toman su forma definitiva en tiempo de ejecución en función del modelo del usuario. Al margen de estas formas adaptativas, PEGASUS admite, aunque no incluye por sí mismo, cualquier otro mecanismo de construcción y modificación dinámica de la estructura del curso. El sistema se ocupa de cómo esto puede afectar a la presentación, pero cómo se actualizan la estructura y el estado del curso es externo al sistema de presentación.

4. Modelo de la presentación

Los sistemas hipermedia adaptativos existentes carecen de un modelo explícito de la presentación. Como consecuencia, la presentación está en parte entremezclada con los contenidos [DeBra98], y en parte es construida automáticamente por el sistema de acuerdo con decisiones de diseño rígidas que el diseñador no puede configurar [Brusilovsky98a, Carro99]. La modelización explícita de la presentación ha sido utilizada en el desarrollo de interfaces de usuario basadas en librerías de componentes interactivas [Castells97]. Con PEGASUS hemos tomado algunas de estas ideas para llevarlas al entorno web, aplicándolas al desarrollo de sistemas hipermedia adaptativos, caracterizados por el volumen, riqueza y complejidad del conocimiento relativo al dominio, que ocupa un primer plano en estos sistemas.

En PEGASUS, la separación entre contenidos y presentación se consigue mediante la definición de una *plantilla de presentación* para cada clase de la ontología. Las plantillas determinan qué partes (atributos y relaciones) de un objeto del dominio deben ser incluidas en su presentación y en qué orden, su apariencia visual, y la disposición espacial. Las plantillas se complementan con *reglas de presentación*, que se ocupan de generar elementos de presentación adaptativa que involucran relaciones entre objetos del dominio, a partir de descripciones de alto nivel muy concisas dadas en las plantillas. Así como en Eon [Murray98] las interfaces de usuario del tutor se asocian con contenidos concretos, en PEGASUS se definen presentaciones para *categorías* de conocimiento. PEGASUS utiliza un modelo de la presentación modular, descentralizado, y adaptado a un modelo del conocimiento, a diferencia de [Saiz98], donde la generación de páginas tiene lugar tomando como eje una única plantilla, que extrae la información necesaria de una base de datos.

4.1. Plantillas

Las plantillas se definen mediante una extensión de HTML basada en JavaServer Pages (JSP) [Sun01], que permite intercalar sentencias de control (entre `<%` y `%>`) y expresiones Java (entre `<%=` y `%>`) en el código HTML. En estas plantillas 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, los elementos del dominio a presentar. Por ejemplo, una plantilla sencilla para la clase Algoritmo podría ser la siguiente:

```
<h2> <%= title %> </h2>
<h3> Previous concepts </h3>
<%= prerequisites %>
<h3> Procedure </h3>
<%= procedure %>
<h3> Examples </h3>                    (4)
<%= examples %>                        (5)
<h3> Proof of Correction </h3>         (6)
<%= correction %>                      (7)
```

En estas plantillas el autor de la presentación sólo tiene que referenciar atributos y relaciones de la clase presentada (en el ejemplo, en negrita). El sistema de presentación se ocupa internamente de aspectos como el manejo automático de listas (relaciones multivaluadas, p.e. los ejemplos del algoritmo), o la aplicación recursiva de plantillas a los objetos referenciados según su clase (p.e. los teoremas de corrección del algoritmo). La página resultante para el algoritmo de Dijkstra con esta plantilla de presentación se puede ver en la Figura 2. Los elementos HTML que rodean a la presentación del algoritmo (estructura de frames con un índice contextual a la izquierda y botones *Previous* / *Next* al pie) corresponden a la plantilla para la clase raíz *DomainObject*.

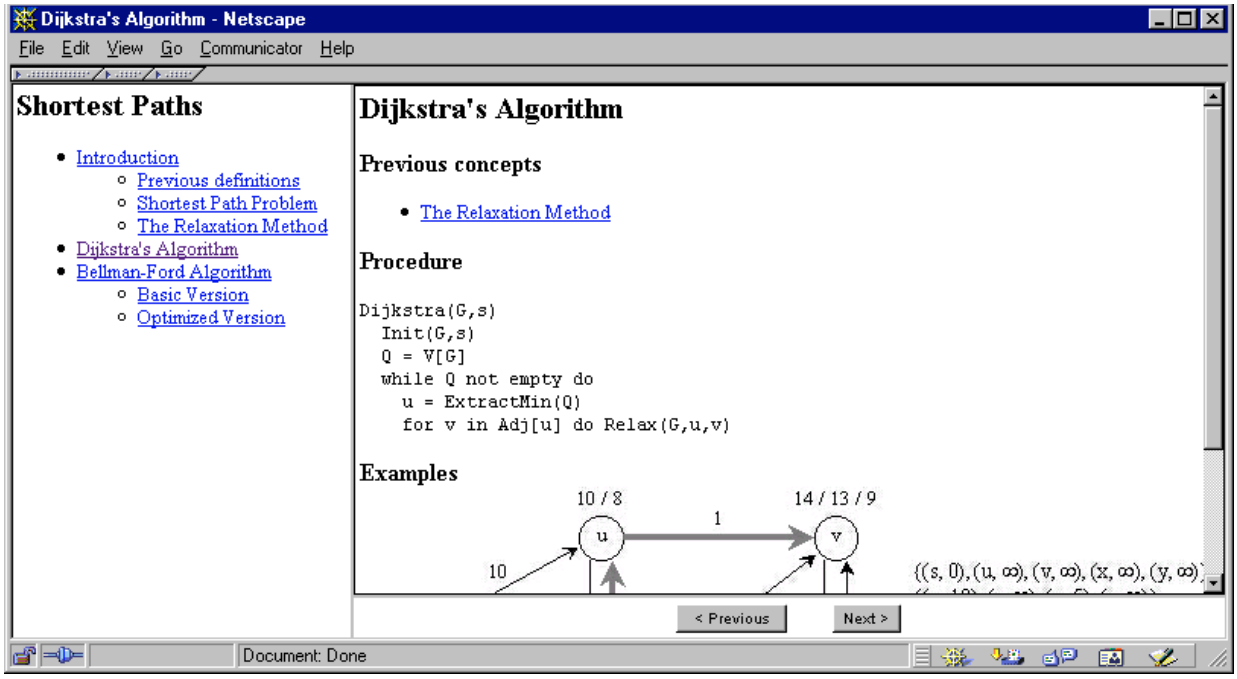

Figura 2. Página web generada para un tópico de tipo Algoritmo

El lenguaje de definición de plantillas permite introducir elementos adaptativos mediante el uso de condicionales. Así, en las líneas 4 a 7 del ejemplo anterior se podría condicionar la información presentada al nivel de experiencia del alumno, incluyendo todos los ejemplos disponibles cuando el alumno es principiante, y un sólo ejemplo, para alumnos más avanzados, mostrando la prueba de corrección sólo si es relevante y su dificultad no es excesiva para el alumno:

```
<% if (user.expertise < 0.5) { %>
     <%= examples %>
<% } %>
<% else { %>
     <%= examples.upto(1) %>
<% } %>
<% if (proof.relevant &&
       proof.difficulty <
          user.expertise) { %>
     <h3> Proof of Correction </h3>
     <%= correction %>
<% } %>
```

El lenguaje de las expresiones incluye así mismo facilidades para, por ejemplo, recortar, filtrar u ordenar listas de acuerdo con una función de comparación arbitraria, generar árboles y listas encadenadas recorriendo una relación, o forzar la generación de enlaces. El uso básico del lenguaje de las plantillas permite especificar un amplio conjunto de presentaciones no triviales mediante una sintaxis muy sencilla. Sin embargo el diseñador puede complicar el lenguaje tanto como quiera escribiendo código Java arbitrario dentro de las propias plantillas.

4.2. Reglas

Las reglas de presentación gobiernan aspectos como la generación de enlaces, la correspondencia entre estilos de enlace y estados de objetos del dominio, la ordenación y disposición espacial de listas (de enlaces o fragmentos), y la generación de presentaciones predefinidas para subconjuntos de la red semántica como árboles y listas encadenadas. Cuando, como en el apartado anterior, el diseñador hace referencia a una relación como *prerequisite* en la plantilla de la clase Algoritmo, las reglas se ocupan automáticamente de decidir si se insertan los detalles de cada prerequisito en la página generada, o si se genera un enlace para cada uno, qué estilo y anotación se utiliza en este caso, y cómo todas las piezas se colocan espacialmente. En este proceso, se analiza si la relación es simple o múltiple, la clase de los tópicos o fragmentos involucrados, su estado, y otras condiciones, en su caso, establecidas por el diseñador. El diseñador puede modificar las reglas existentes o definir las suyas propias.

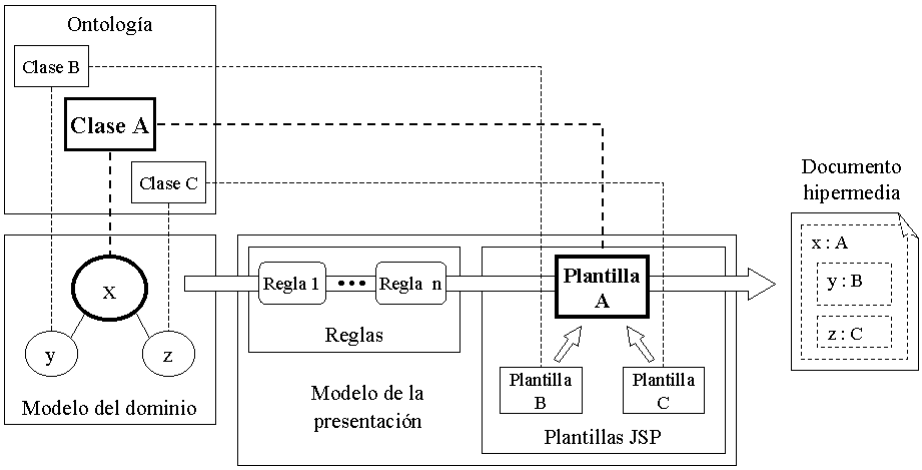

Figura 3. Generación de documentos web para unidades de conocimiento utilizando plantillas y reglas

Las reglas consisten en una lista de cero o más condiciones, seguidas por la presentación a aplicar cuando éstas se cumplen, descrita con la misma sintaxis de las plantillas. Por ejemplo, la siguiente regla establece un determinado color para los enlaces a objetos (unidades de conocimiento) que ya han sido leídos y cuyos prerequisitos son todos conocidos por el usuario.

```
<Rule class="DomainObject">
  <test condition="asLink && read"/>
  <test-every var="item"
              list="prerequisites"
              condition="item.known"/>
  <presentation>
     <font color="#006600">
        <%= this %>                    (8)
     </font>
  </presentation>
</Rule>
```

Para que las listas de enlaces se muestren mediante una lista HTML de tipo `<ul>` se puede definir una regla como la siguiente.

```
<Rule class="List[DomainObject]">
  <test-every var="item" list="this"
              condition="item.asLink"/>
  <presentation>
    <ul>
      <iterate var="item" list="this">
        <li> <%= item %> </li>         (9)
      </iterate>
    </ul>
  </presentation>
</Rule>
```

Si bien escribir reglas es una tarea delicada que requiere una familiaridad con el sistema, cualquier autor con conocimientos básicos de HTML podría retocar una regla como la anterior para utilizar, por ejemplo, tablas en lugar de listas HTML.

Cuando el sistema de presentación recibe la instrucción de presentar un objeto (como "x" en la Figura 3), antes de aplicar la plantilla correspondiente PEGASUS intenta aplicar todas las reglas existentes. Cuando en la parte derecha de una regla aparecen referencias a objetos (como parte de una relación, o explícitamente como en las expresiones `<%= this %>` y `<%= item %>` de las líneas 8 y 9 más arriba), éstos son procesados a su vez, aplicando nuevamente las reglas. Cuando ya no es posible aplicar más reglas, se aplica la plantilla que corresponde al objeto según su clase ("A" en la figura); en este sentido, las plantillas se pueden ver como reglas de mínima prioridad cuya condición es *true*. El procedimiento se repite recursivamente con los objetos que a su vez aparezcan al procesar la plantilla ("y", "z" en la figura).

5. Arquitectura

En tiempo de ejecución, el estudiante interactúa con la aplicación desde un navegador web. La interacción con una aplicación construida con PEGASUS se traduce en un recorrido por la red de objetos del dominio. Cada vez que el usuario se desplaza a un objeto, PEGASUS responde generando una página HTML. Para ello, el sistema 1) resuelve la petición del usuario determinando el objeto al que se debe ir, 2) localiza la instancia en el modelo del dominio, 3) actualiza el modelo del dominio y del usuario, 4) genera la presentación HTML aplicando las reglas pertinentes y la plantilla correspondiente a su clase (ver Figura 4). En las páginas generadas, los enlaces no corresponden a otras páginas, sino que designan, de forma explícita o descriptiva, objetos del dominio.

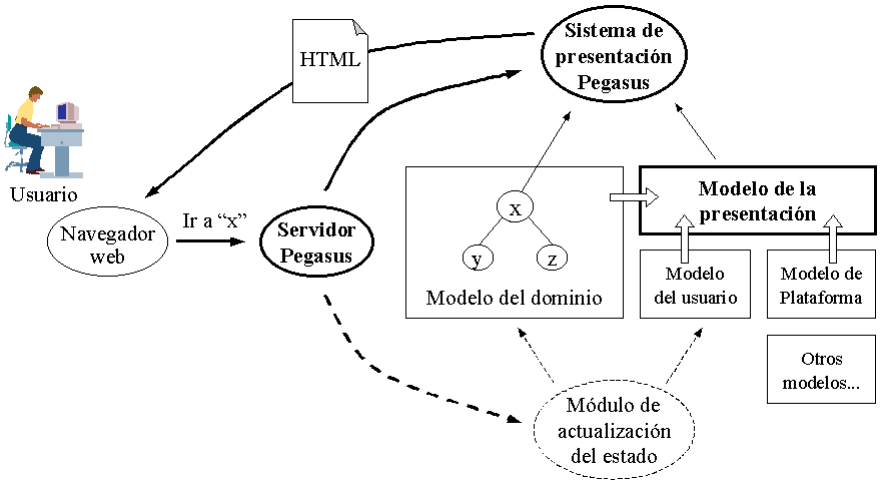

Figura 4. Arquitectura del sistema

En nuestro sistema, la unidad de interacción con el usuario es pues la petición HTTP. La actualización del modelo del usuario se lleva a cabo teniendo en cuenta únicamente la información transmitida en estas peticiones. Las características de la plataforma e interfaz de usuario se capturan en el cliente mediante código JavaScript que PEGASUS inserta en las páginas HTML generadas, y esta información es enviada al servidor como parte de la petición HTTP cuando el usuario pulsa sobre enlaces y botones de las páginas. Esta suposición simplifica sensiblemente la arquitectura del sistema y la integración con módulos y herramientas externos. A cambio, supone que el sistema no tiene información inmediata y detallada de la actividad del usuario entre dos peticiones, y tampoco se actualiza la presentación en este intervalo. Una granularidad más fina en la interacción se podría soportar generando componentes de interfaz de usuario en Java (applets) que interaccionen con el usuario y se comuniquen directamente con el servidor para consultar y actualizar los modelos del dominio y del usuario.

En general nuestro sistema de presentación no funciona sólo, ya que los pasos 1 y 3 descritos anteriormente son externos a PEGASUS. Una vez que se ha definido una ontología para la materia, se necesita un módulo que construya y/o actualice las redes de tópicos en tiempo de ejecución (paso 3), como se muestra en la Figura 4, aplicando estrategias pedagógicas como las que utilizan las herramientas descritas en la Sección 2. Opcionalmente se puede incluir un sistema de planificación como en DCG [Vassileva97], para determinar el camino a seguir en respuesta a las peticiones del usuario (paso 1).

6. Conclusiones

El sistema de representación del conocimiento propuesto permite reproducir los modelos del dominio utilizados en una amplia gama de sistemas hipermedia. La generación dinámica de la presentación independientemente de la actualización del estado de la aplicación hace que PEGASUS sea compatible con herramientas de soporte para cursos adaptativos como las descritas en la sección de antecedentes, pudiendo ser utilizado como un módulo complementario de estas herramientas. Nuestro enfoque permite la especificación de la presentación separadamente de la construcción de contenidos, favoreciendo la reutilización y consistencia de la presentación, reduciendo por tanto el coste de desarrollo.

Si bien la construcción de plantillas está al alcance de cualquier diseñador de páginas web familiarizado con las tecnologías HTML y JSP, la definición de ontologías es una tarea delicada que requiere la participación de un diseñador avanzado, entrenado en el uso de la herramienta. Una vez definida la ontología y los modelos de presentación asociados, la construcción del modelo del dominio es asequible para el autor medio. La introducción de modificaciones sobre las plantillas y reglas de presentación puede ser un paso fácil para este tipo de autor hacia un nivel de uso más avanzado.

Actualmente existe una implementación completa de PEGASUS sobre Java (JDK 1.3), con XML/DOM y JavaServer Pages [Sun01]. Hemos desarrollado así mismo una herramienta de autor interactiva para facilitar la construcción de las ontologías y las redes de objetos del dominio

[Macías01c]. Nuestros planes para el futuro inmediato incluyen el desarrollo de herramientas de autor que permitan diseñar el modelo de la presentación mediante ejemplos, editando las páginas HTML generadas a partir de un modelo inicial, como en [Castells99]. La creación de este tipo de herramientas no es posible sin un modelo declarativo explícito de la presentación.

Agradecimientos

El trabajo presentado en este artículo está parcialmente financiado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), proyecto nº TEL1999-0181.

Referencias

[Brusilovsky98a] 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.

[Brusilovsky98b] Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. En: Brusilovsky, P., Kobsa, A., Vassileva, J. (eds.): Adaptive Hypertext and Hypermedia. Kluwer Academic Publishers, 1998, 1-43.

[Carro99] Carro, R. M., Pulido, E., Rodríguez, P.: Dynamic generation of adaptive Internet-based courses. Journal of Network and Computer Applications 22, 1999, pp. 249-257.

[Castells97] Castells, P., Szekely, P., and Salcher, E.: Declarative Models of Presentation. Actas International Conference on Intelligent User Interfaces (IUI'97), Orlando, Florida, 1997, pp. 137-144.

[Castells99] 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.

[Castells01] Castells P., Macías, J. A.: An Adaptive Hypermedia Presentation Modeling System for Custom Knowledge Representations. Actas World Conference on the WWW and Internet (WebNet'2001). Orlando, Florida, USA, 2001, pp. 148-153.

[daSilva98] da Silva, D. P., Van Durm, R., Duval, E., Olivié, H.: Concepts and Documents for Adaptive Educational Hypermedia: a Model and a Prototype. Actas 2nd Workshop on Adaptive Hypertext and Hypermedia (HYPERTEXT'98). Pittsburg, USA, 1998.

[DeBra98] De Bra, P., Calvi, L.: AHA! An open Adaptive Hypermedia Architecture. The New Review of Hypermedia and Multimedia, 4. Taylor Graham Publishers, 1998, pp. 115-139.

[Fensel99] Fensel, D., Angele, J., Decker S., Erdmann M., Schnurr, H. P., Staab, S., Studer, R., and Witt, A.: On2broker: Semantic-based access to information sources at the WWW. Actas World Conference on the WWW and Internet (WebNet'99). Honolulu, Hawaii, 1999, pp. 366-371.

[Gruber93] Gruber, T. R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In Guarino, N. and Poli, R. (eds.), Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Press, Boston, 1993.

[Macías01a] Macías, J. A., Castells, P.: A Generic Presentation Modeling System for Adaptive Web-based Instructional Applications. Actas ACM Conference on Human Factors in Computing Systems (CHI'2001), Extended Abstracts. Seattle, Washington, 2001, pp. 349-350.

[Macías01b] Macías, J. A., Castells, P.: Adaptive Hypermedia Presentation Modeling for Domain Ontologies. Actas 10th International Conference on Human-Computer Interaction (HCII '2001). New Orleans, Louisiana, 2001, pp. 710-714.

[Macías01c] 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, pp. 268-278.

[Murray98] Murray, T.: Authoring Kowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. Journal of the Learning Sciences 7, 1, 1998, pp. 5-64.

[Pérez95] Pérez, T. A., Gutiérrez, J., Lopistéguy, P.: An Adaptive Hypermedia System. Actas Artificial Intelligence in Education (AIED'95). AACE, Charlottesville, 1995.

[Saiz98] Saiz, F., Szekely, P., Devang, D., Customized Web-based Data Presentation. Actas World Conference on the WWW, Internet and Intranet (WebNet'98), Orlando, Florida, 1998, pp. 792-798.

[Sun01] Sun MicroSystems, Inc.: JavaServer Pages 1.2 Draft Specification, 2001. Disponible en la Web: http://java.sun.com/products/jsp.

[Trigg86] Trigg, R. H., Weiser, M.: Textnet: A Network-based Approach to Text Handling.

ACM Transactions on Office Information Systems (TOIS) 4, 1, January 1986, pp. 1-23.

[Vassileva95] Vassileva, J.: Dynamic Courseware Generation: at the Cross Point of CAL, ITS and Authoring. Actas International Conference on Computers in Educaiton (ICCE'95). AACE, Singapoore, 1995, pp. 290-297.

[Vassileva97] Vassileva, J.: Dynamic Course Generation on the WWW. Actas 8th World Conference on Artificial Intelligence in Education (AIED'97). Kobe, Japón, 1997, pp. 498-505.

[Weber97] Weber, G., Specht, M.: User modeling and Adaptive Navigation Support in WWW-based Tutoring Systems. Actas 6th International Conference on User Modeling (UM97), Sardinia, Italia, 1997.

[Wilkinson99] Wilkinson, R., Smeaton, A. F.: Automatic Link Generation. ACM Computing Surveys, 31, 4es, December 1999.