

MODELOS HIPERMEDIA

Toni Navarrete Terrasa

**Trabajo del curso *Guió d'Interactius*
Profesor: Xavier Berenguer
Doctorado en Informàtica i Comunicaci3n Digital
Universitat Pompeu Fabra
Verano del 2000**

Índice

1. INTRODUCCIÓN.....	2
2. MODELO DE HIPERTEXTO DE DEXTER.....	3
2.1. BREVE DESCRIPCIÓN DEL SISTEMA.....	3
2.2. MODELO SIMPLE DE LA CAPA DE ALMACENAMIENTO.....	5
2.2.1. <i>Enlaces span-to-span</i>	6
2.3. CAPA DE TIEMPO DE EJECUCIÓN.....	9
2.4. CONCLUSIONES.....	11
3. MODELO DE HIPERMEDIA DE AMSTERDAM.....	12
3.1. DESCRIPCIÓN DE LOS CONCEPTOS DEL AHM.....	12
3.1.1. <i>Hipertexto, multimedia e hipermedia</i>	12
3.1.2. <i>Información temporal</i>	14
3.1.3. <i>Enlaces en hipermedia</i>	16
3.1.4. <i>Atributos de especificación de la presentación a alto nivel</i>	16
3.2. EL MODELO DE HIPERMEDIA DE AMSTERDAM.....	17
3.3. RELACIONES TEMPORALES.....	19
3.4. OTRAS CONTRIBUCIONES DEL MODELO.....	19
3.4.1. <i>Contexto de los enlaces</i>	19
3.4.2. <i>Canales</i>	20
3.5. CONCLUSIONES.....	20
4. CONCLUSIONES FINALES.....	21
5. REFERENCIAS BIBLIOGRÁFICAS.....	22

1. Introducción

A finales de la década de los 80 surge la necesidad de especificar qué es el, en ese momento, nuevo campo del hipertexto/hipermedia y definir toda una serie de conceptos asociados como por ejemplo enlace, navegación,...

En el presente trabajo pretendo presentar los dos primeros modelos que se crearon a tal efecto y que pese al tiempo transcurrido son perfectamente válidos hoy en día. En primer lugar analizaremos el modelo de Dexter, el primer intento, que data de finales de los 80, de investigar las características propias de la hipermedia, ofreciendo un modelo que describe su estructura. Dexter proporciona también una terminología que a partir de entonces se consideraría estándar en este campo. Posteriormente se analiza el modelo de Amsterdam, fuertemente basado en el anterior, al que se añade el concepto de tiempo, que Dexter no consideró, principalmente debido a que Dexter se ciñe fundamentalmente a hipertexto, y no considera la variedad de medios existentes y el principal problema que conllevan, que son las relaciones temporales.

Este trabajo está basado en gran medida en mi proyecto final de carrera de la Ingeniería en Informática y que llevó por título "Una metodología relacional hipermedia. Estudio en casos prácticos" [1].

2. Modelo de Hipertexto de Dexter

El modelo de referencia de hipertexto de Dexter surge en 1988 durante unas reuniones, la primera de las cuales tuvo lugar en la Dexter Inn (New Hampshire), de donde toma su nombre, como un intento de respuesta a la pregunta: ¿qué tienen en común los distintos sistemas de hipertexto en sus nociones de nodos y enlaces (*links*) entre dichos nodos?

Así, se pretende que el modelo sirva como un estándar para comparar las características y funcionalidades de varios sistemas de hipertexto. También como una base sobre la cual desarrollar estándares para interoperabilidad e intercambio entre sistemas de hipertexto.

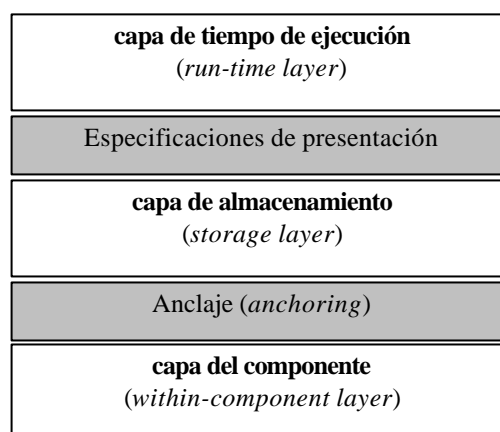
Otro punto importante de este trabajo es el de proveer de una terminología común al campo del hipertexto.

Nótese que, a pesar de que a menudo se diferencia entre hipertexto e hipermedia, aquí los autores no hacen tal distinción.

Las presentes notas están basadas en el artículo [HAL94].

2.1. Breve descripción del sistema

El modelo de Dexter divide el sistema hipertexto en tres capas diferentes. La primera, la más próxima al usuario es la denominada **capa de tiempo de ejecución** (*run-time layer*); la intermedia es la llamada **capa de almacenamiento** (*storage layer*), mientras que la última recibe el nombre de **capa del componente** (*within-component layer*).



El modelo se centra en la capa de almacenamiento, que modela la estructura básica de nodos y enlaces. Vendría a ser como una base de datos donde se almacena una jerarquía de **componentes** (contenedores de la información) interconectados por enlaces.

Los componentes es lo que conocemos típicamente como nodos (por ejemplo una tarjeta en Hypercard, o un *frame* en Director). Por tanto, un componente puede contener fragmentos de texto, gráficos, animaciones, ... Al estudiar la capa de almacenamiento veremos que éstos son lo que llamamos componentes atómicos, que al unirse forman componentes compuestos, que es el equivalente a la noción de nodo.

La capa de almacenamiento estudia los mecanismos por los cuales se unen dichos componentes y enlaces.

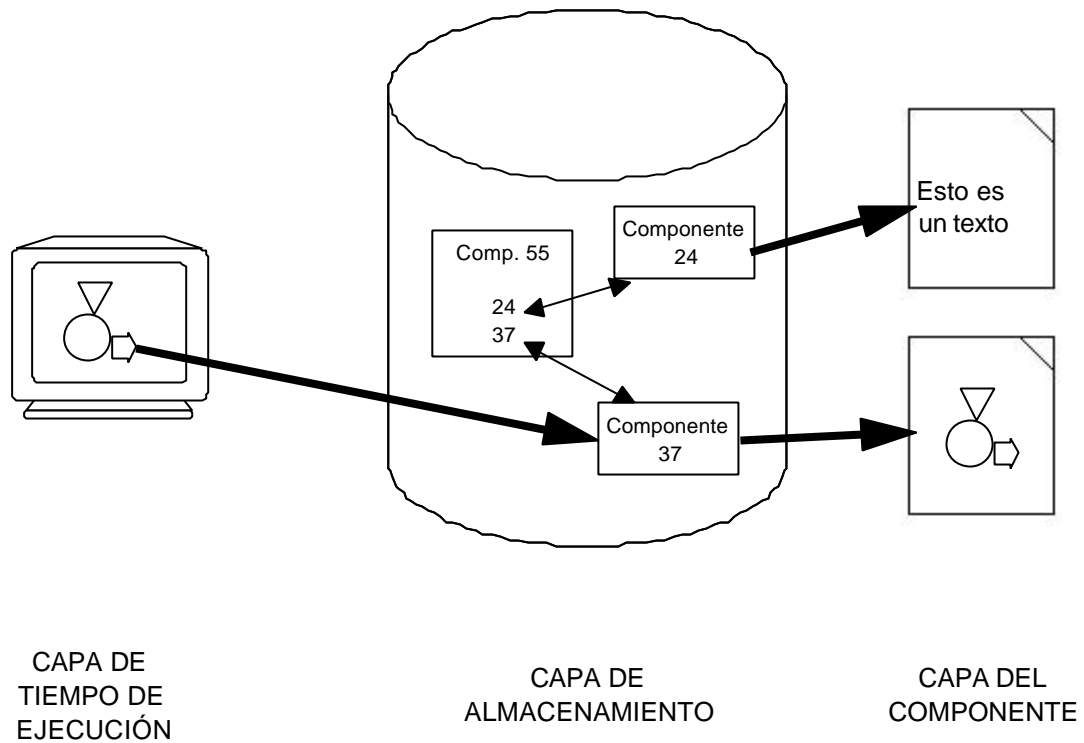
Por contra, la capa del componente refleja los contenidos y estructura dentro de los componentes. Debido a que hay un sin fin de posibles contenidos/estructuras que pueden ser incluidos en un componente, el modelo de Dexter no profundiza el estudio de esta capa, dando libertad en este nivel.

Una parte fundamental del modelo de Dexter es la interfaz entre las capa de almacenamiento y la capa del componente. Se basa en el mecanismo de **anclaje** (*anchoring*), que veremos más adelante.

En cuanto a la capa de tiempo de ejecución, ésta provee al usuario de unas herramientas para acceder, ver y manipular la estructura de la red de hipertexto. Al igual que en la capa del componente, también existen un gran número de posibles herramientas para acceder, ver y manipular un hipertexto. Por consiguiente, el modelo Dexter sólo ofrece unas ideas básicas de cómo son esos mecanismos, pero no puede hacer referencia a los detalles de cómo el usuario interactúa con el hipertexto.

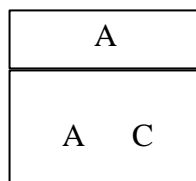
Un punto importante es la interfaz entre esta capa y la capa de almacenamiento, a la que llama **especificaciones de presentación** (*presentation specifications*). Ésta es un mecanismo que permite que la forma en que un componente se presenta al usuario pueda estar en función no sólo de una herramienta de hipertexto específica (es decir, de una capa de tiempo de ejecución específica), sino que puede ser una propiedad del componente por sí mismo, del enlace tomado para llegar a tal componente o también de las preferencias del usuario. Por ejemplo, un componente que contiene una animación puede presentarse de dos maneras diferentes según el usuario. Si el usuario es un profesor, se le permitirá editar el componente (la animación), mientras que si el usuario es un alumno, sólo se le permite la visualización.

En la siguiente figura, vemos una representación más gráfica de las tres capas del modelo de Dexter. En la capa de tiempo de ejecución, tenemos la presentación del hipertexto al usuario a través de la pantalla. En la capa de almacenamiento, vemos una base de datos donde se almacenan todos los componentes y enlaces entre ellos (los enlaces también son componentes). Y por último en la capa del componente, se guarda la estructura interna del componente.

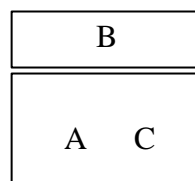
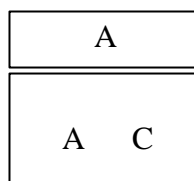


2.2. Modelo simple de la capa de almacenamiento

La capa de almacenamiento describe la estructura de un conjunto finito de componentes, junto con dos funciones, *resolver* y *accessor*, encargadas de recuperar componentes. Es el componente el elemento fundamental de esta capa; éste puede ser un átomo, un enlace entre componentes, o puede estar compuesto a partir de otros componentes (formando un grafo acíclico, ya que un ciclo implicaría una recursividad). En el siguiente ejemplo vemos dos casos prohibidos de componentes compuestos, debido a que crean recursividad. Cada cuadro representa un componente compuesto; en la parte superior aparece en nombre del componente y en la inferior los componentes (atómicos o compuestos) que la forman.



Ciclo: A no puede contener a A



Ciclo: A no puede contener a un componente que contiene a

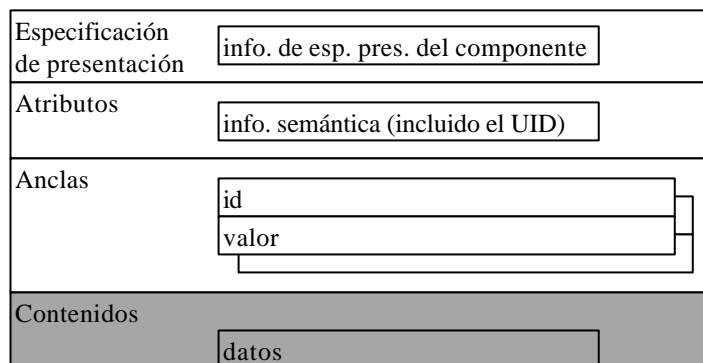
Cada componente tiene un identificador único (UID). La función *accessor* es la que accede a un componente dado su UID, mientras que la función *resolver* permite, a partir de unas especificaciones del componente recuperar los UID correspondientes. Después se podrá acceder al componente mediante *accessor*. Por tanto, *accessor* constituye un direccionamiento directo a las componentes, mientras que *resolver* se utiliza para un direccionamiento indirecto, a dichos componentes a partir de las especificaciones.

2.2.1. Enlaces *span-to-span*

Los enlaces *span-to-span* son aquellos que van de una parte de una componente a otra parte de otro componente.

Para conseguir los enlaces *span-to-span* Dexter provee de una entidad de direccionamiento indirecto llamada **ancla** (*anchor*). Un ancla tiene dos partes: un identificador (*id ancla - anchor id*) y un valor (*anchor value*). El identificador es único dentro de cada componente. Por tanto, el par UID componente - *id ancla* es único dentro del hipertexto. El *id ancla* es manejado por la capa de almacenamiento para referenciar al contenido del componente, y se mantiene siempre constante; mientras, el valor del ancla es utilizado sólo en la capa del componente, y cuando se edita y modifica el componente. Por otro lado, el valor del ancla especifica de forma arbitraria alguna localización, región, ítem o subestructura dentro de componentes. Éste es sólo interpretable por las aplicaciones responsables de manejar el contenido/estructura del componente.

Veamos cómo resulta la estructura de un componente atómico, teniendo en cuenta las anclas:



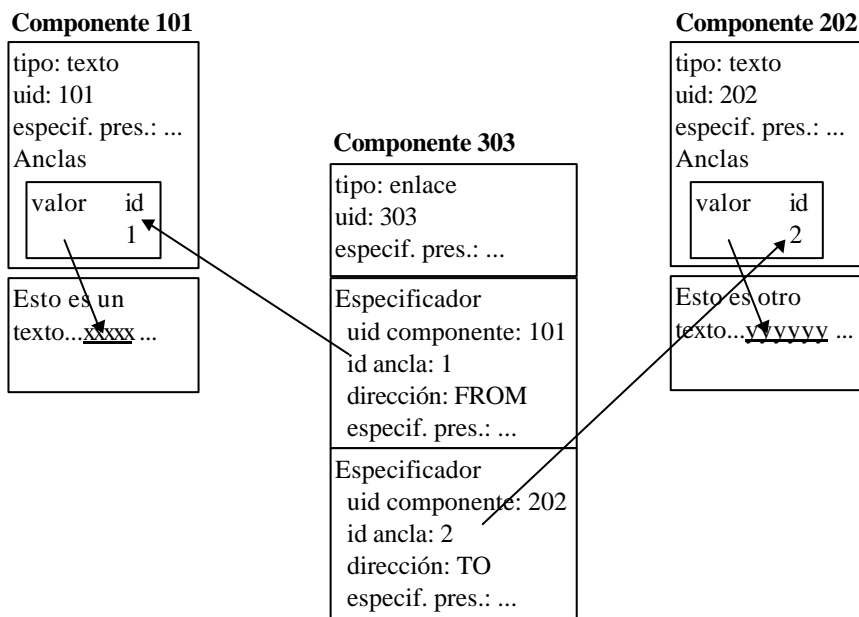
Y en el caso de un componente compuesto, la única diferencia es que se guardan también los UIDs de los componentes hijos

Especificación de presentación	info. de esp. pres. del componente			
Atributos	info. semántica (incluido el UID)			
Anclas	<table border="1"> <tr> <td>id</td> <td rowspan="2">}</td> </tr> <tr> <td>valor</td> </tr> </table>	id	}	valor
id	}			
valor				
Hijos	<table border="1"> <tr> <td>UID componente hijo</td> <td rowspan="2">}</td> </tr> <tr> <td></td> </tr> </table>	UID componente hijo	}	
UID componente hijo	}			
Contenidos	datos			

A partir de aquí, para representar los extremos de un enlace, emplearemos el id ancla combinado con la especificación del componente, de cada extremo. Es lo que llamamos un especificador (*specifier*). Además de la especificación de un componente y un id ancla, son utilizados dos campos más: una dirección y una especificación de presentación de ese enlace. Como vemos, el especificador constituye, un extremo del enlace, y por tanto, un enlace está formado por dos o incluso más especificadores. En el caso de tener más de dos especificadores, tenemos lo que se denominan enlaces múltiples (*multiway links*). El campo dirección puede tomar varios valores: TO, FROM, BIDIRECT y NONE; el campo especificación de presentación se refiere a la interfaz entre la capa de almacenamiento y la capa de tiempo de ejecución, de la cual se hablará posteriormente más en detalle.

A continuación se muestra un ejemplo de enlace entre los componentes atómicos 101 y 202. Cada uno de ellos tiene definido un ancla que apunta a un punto concreto del texto (valor) y con un identificador. El componente 303 es un enlace entre los citados componentes. Por tanto, tiene dos especificadores. Recordemos que los especificadores se definen con una pareja UID componente - id ancla. Por tanto, el primero tiene el par (101,1) y el segundo (202,1). Además, de esta pareja, el identificador debe definir el sentido del enlace, en este caso del 101 (FROM) al 202 (TO).

Vemos primero un diagrama y después la descripción escrita del modelo.



```

<hipertexto>
  <componente>
    <tipo>texto</tipo>
    <uid>101</uid>
    <especificación de presentación>...</especificación de presentación>
    <datos>Esto es un texto ...</data>
    <ancla>
      <id>1</id>
      <valor>22-26</valor>      (La posición del ancla en el texto)
    </ancla>
  </componente>
  <componente>
    <tipo>texto</tipo>
    <uid>202</uid>
    <datos>Esto es otro texto ...</data>
    <ancla>
      <id>2</id>
      <valor>35-46</valor>
    </ancla>
  </componente>
  <componente>
    <tipo>enlace</tipo>
    <uid>303</uid>
    <especificación de presentación>...</especificación de presentación>
    <especificador>
      <uid componente>101</id>
      <id ancla>1</valor>
      <dirección>FROM</dirección>
      <especificación de presentación>...</especificación de presentación>
    </especificador>
    <especificador>

```

```
<uid componente>202</id>
<id ancla>2</valor>
<dirección>TO</dirección>
<especificación de presentación>...</especificación de presentación>
</especificador>
</componente>
```

Una serie de funciones nos permiten acceder y modificar la capa de almacenamiento. Estas funciones, sobre las que no profundizaremos, nos permiten, entre otras cosas, crear un componente atómico, crear un componente compuesto, crear un enlace, modificar o borrar los anteriores.

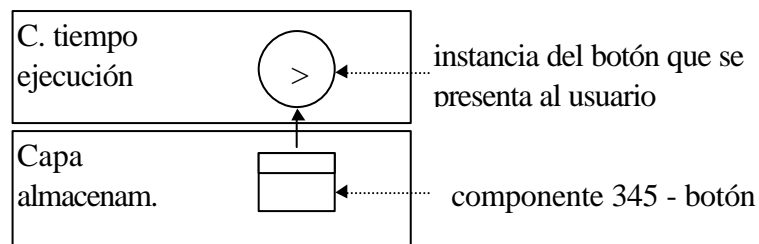
Por último, no deben obviarse nunca estas cinco reglas:

- La función *accessor* debe permitir que cada componente sea accesible desde un UID.
- La función *resolver* debe ser capaz de producir todos los posibles UIDs válidos.
- No hay ciclos en las relaciones componente/subcomponente (un componente no puede ser subcomponente de sí mismo).
- Los identificadores de anclas de un componente han de ser los mismos que los identificadores de anclas del especificador de enlace, cuyo extremo es ese componente.
- El hipertexto debe ser lo que se denomina *link-consistent*, es decir, que al borrar un componente se borren también los enlaces que lo tienen como extremo.

2.3. Capa de tiempo de ejecución

El concepto fundamental de la capa de tiempo de ejecución es la **instanciación** de un componente. Una instanciación es una presentación de un componente al usuario.

El funcionamiento es similar al de una cache: una copia (instanciación) de un componente se pasa al usuario para que la visualice y/o edite. Esta copia será después escrita de nuevo en la capa de almacenamiento. Nótese que puede haber más de una instanciación de un componente a la vez. Por ello, a cada instanciación se le asigna un identificador único (*instantiation identifier* - **IID**).



Recordemos que la manera en que el componente se presenta al usuario dependerá de las preferencias que tenga definidas en las especificaciones de presentación.

La instanciación de un componente también conlleva la de sus anclas. A esto se le llamará **marcador de enlace** (*link marker*), que es la manifestación visible de un ancla en un documento.

En resumen, una instanciación contendrá:

- una instanciación base (*base instantiation*), que es una primitiva del modelo que representa algún tipo de presentación de el componente al usuario.
- una secuencia de marcadores de enlace junto con una función que haga corresponder esos marcadores de enlaces a las anclas que instancian.

Como que en un momento dado un mismo usuario puede estar visualizando y/o editando varias instanciaciones de componentes, se define una nueva entidad en el modelo que es la **sesión**, que permite saber momento a momento la relación entre los componentes y sus instanciaciones. Así, cuando un usuario pretende acceder a un hipertexto, lo que hace es abrir una sesión. Y esto es lo que será necesario para cada sesión:

- el hipertexto que es accedido
- los IID de las instanciaciones a componentes de la capa de almacenamiento que se utilizan en esta sesión
- una historia, que contiene las operaciones que se han hecho desde que se abrió la sesión
- una función *run-time resolver*, que es el equivalente a la función *resolver* de la capa de almacenamiento
- una función de instanciación (*instantiator*), que devolverá, a partir de un UID y una especificación de presentación (otra primitiva del modelo que define cómo se “presentará” un componente), una instanciación del componente como parte de la sesión.

- una función *realizer*, que es la inversa de la función de presentación. Es decir, que a partir de una instancia, devuelve un componente que “refleje” el estado actual de la instanciación. Esto es usado para el mecanismo de “*writing-back in cache*”, es decir, de crear o modificar el componente en la capa de almacenamiento.

Al igual que teníamos un conjunto de funciones que permitían acceder y modificar la capa de almacenamiento, también tenemos un conjunto de operaciones que nos lo permiten sobre la capa de tiempo de ejecución. Estas operaciones, sobre las que tampoco profundizaremos, nos permiten, entre otras cosas, crear una sesión, crear una instanciación para un componente, actuar sobre el componente asociado con la presentación, seguir un enlace, ...

2.4. Conclusiones

Dexter fue el primer intento de modelar el hipertexto y de dar una terminología común para este, por entonces, nuevo campo. No hay que olvidar que sus comienzos datan de 1988, cuando los sistemas de hipertexto-hipermedia estaban muy lejos de lo que son hoy.

De hecho, la principal carencia que se puede observar en este modelo es que, a pesar de que los autores lo recomiendan para tanto hipertexto como hipermedia, no aborda en ningún momento la complejidad de los distintos medios. En realidad, se podría decir que el modelo está fundamentalmente destinado sólo a hipertexto, y la aplicación a la hipermedia es muy difícil. Por ejemplo, el concepto de valor del ancla lo hace en función de la posición del ancla en el texto, pero esto en otro medio no tiene sentido.

En concreto, no tiene en cuenta los aspectos relacionados con el tiempo, algo fundamental en el audio y el vídeo. Es por ello que posteriormente apareció el modelo de Amsterdam, que basado en Dexter, añade el estudio del tiempo.

Pero, a pesar de eso (no hay que olvidar que estos elementos eran imposibles de utilizar en 1988), el modelo especifica, por primera vez y muy detalladamente, cómo es la estructura interna de un hipertexto. Incluso define unas estructuras que no habían sido utilizadas hasta entonces como los enlaces múltiples, la posibilidad de hacer un enlace a otro enlace (recordemos que los enlaces son componentes) o los componentes compuestos, que por aquel entonces muchos sistemas de hipertexto no soportaban.

3. Modelo de hipermedia de Amsterdam

Hipermedia es la suma de hipertexto y multimedia. El hipertexto nos provee de una estructura de navegación a través de los datos, mientras que la multimedia nos ofrece como punto fundamental una gran riqueza de tipos de datos.

Sin embargo, el modelo de Dexter, aunque según sus autores está orientado a hipermedia, la realidad es que se centra por completo en el hipertexto, dejando de lado el estudio de la variedad de tipos posibles. Y entre esos tipos, no hay que olvidar que los hay dependientes del tiempo. Por tanto, el modelo de Dexter comete un gran olvido: no toma en consideración el concepto del tiempo. Se hace necesario un nuevo modelo de datos multimedia, en el que se van a ver reflejadas relaciones complejas en tiempo. Por ejemplo, es necesario contemplar algo tan típico en una aplicación hipermedia como que a los 4 seg. de que se active un vídeo, aparezca cierto texto (sincronización entre componentes). Sobre estos conceptos surge el Modelo de Hipermedia de Amsterdam (AHM).

En resumen, mientras que el modelo de Dexter permite la composición de estructuras jerárquicas, la especificación de enlaces entre componentes y el uso de anclas, el modelo de Amsterdam lo extiende añadiendo las nociones de tiempo, presentación a alto nivel de atributos y enlaces de contexto.

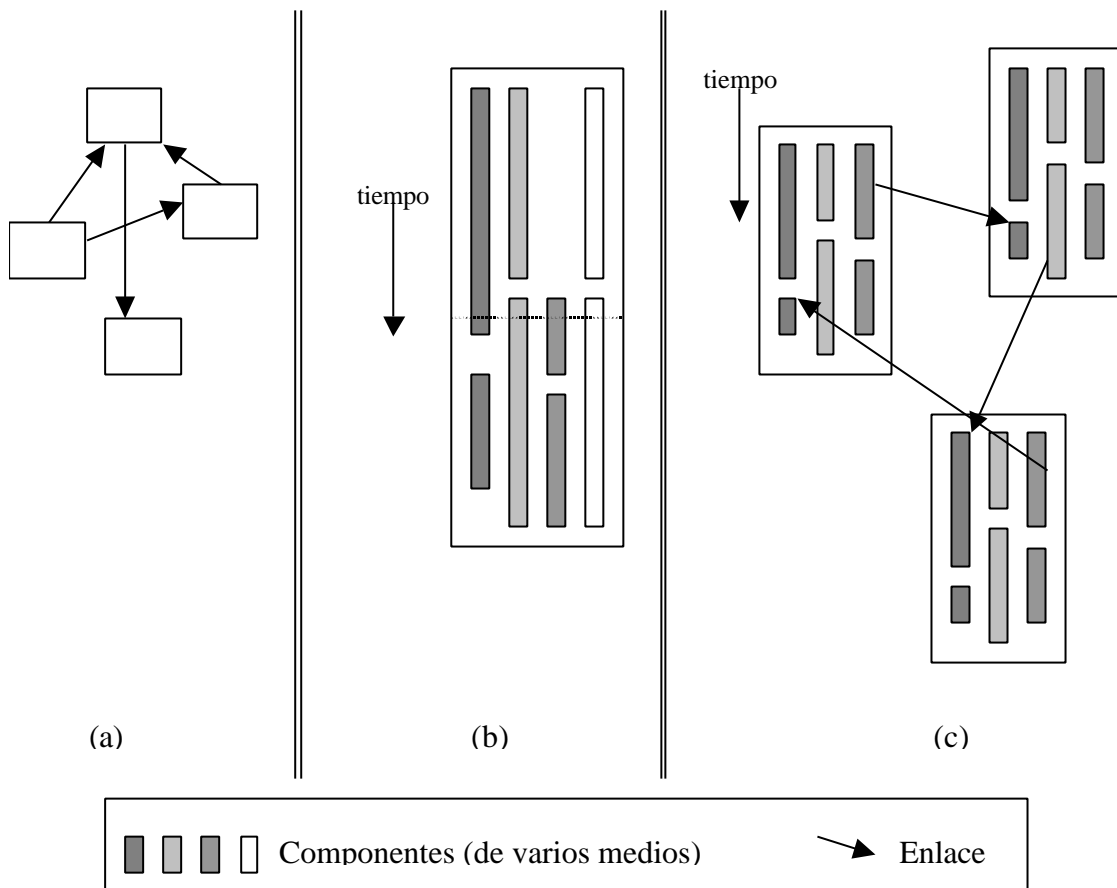
Hay cierto vocabulario ligeramente diferente al empleado en Dexter: un **documento** es una colección completa de componentes, cada uno de los cuales puede estar compuesto de otros componentes, o bien de componentes atómicos, llamados aquí elementos de datos (*data elements*) o también **entidades**. Una **presentación** es la forma activa de un documento. A menudo, se habla indistintamente de documento o de presentación.

Este apartado está basado en el artículo [HAR94].

3.1. Descripción de los conceptos del AHM

3.1.1. Hipertexto, multimedia e hipermedia

Para hablar de un sistema multimedia o hipermedia no es suficiente con tener medios dinámicos que dependan del tiempo. El concepto fundamental es la sincronización entre ellos.



En la parte (a) de la figura, tenemos una red de hipertexto con enlaces entre componentes. La visita a un componente termina bien por la finalización de la aplicación o bien por el paso a otro componente a través de un enlace. Aquí el tiempo apenas cuenta y tenemos básicamente hipertexto.

En la parte (b) de la figura, tenemos una presentación multimedia. El usuario controla qué componente quiere visitar, pero ese componente puede ser dinámico, es decir que varía sin la intervención del usuario, según una noción temporal. Dos tipos de control del usuario sobre las presentaciones se permiten aquí. La primera, la típica similar al panel de control de un cassette o vídeo-reproductor, con los controles *stop/play/fast forward/rewind*. La segunda sería similar a un enlace hipertextual y nos llevaría de un punto temporal del componente a otro (sería similar a un *fast forward* pero parando en un punto definido de antemano).

La parte (c) de la figura muestra la combinación entre hipertexto y multimedia: cada componente de la red hipertexto es una presentación multimedia. Así, hay dos aspectos a tener en cuenta, que son la navegación hiperestructurada dentro del documento y la presentación de la información multimedia. Esto sería hipermedia específicamente.

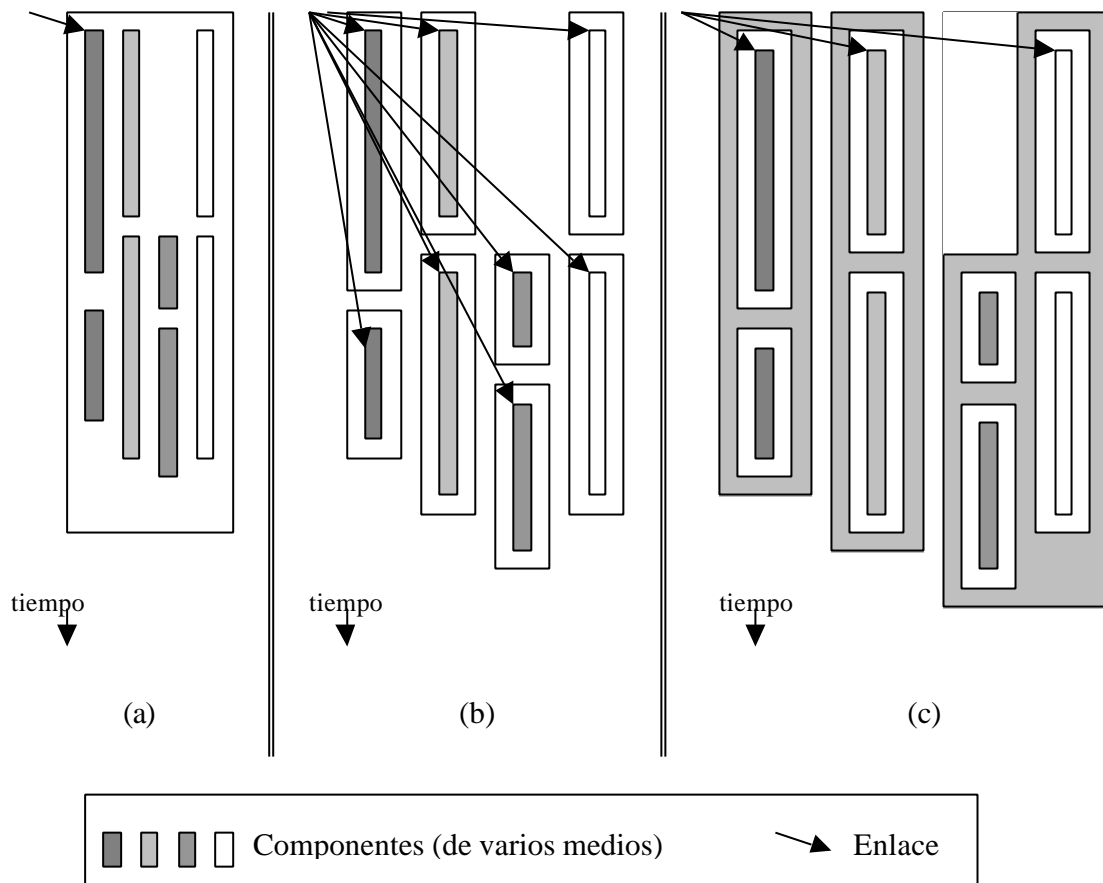
3.1.2. Información temporal

En general, un modelo de hipermedia debería ser capaz de especificar cómo las partes individuales se relacionan unas con otras. Una de esas relaciones es evidentemente la temporal. De hecho, a menudo, estas relaciones temporales se han representado como un simple enlace hipertextual. Esto es conceptualmente un error, ya que así se mezclan dos aspectos separados de una presentación como son las relaciones hipertextuales y las temporales. La solución que se propone en Amsterdam es la de particionar las relaciones temporales entre datos en dos clases:

- aquellas que se refieren a la identificación de los componentes que estarán presentes conjuntamente. A esta clase se la denomina **colección**.
- aquellas que se refieren al orden relativo en el que estos componentes son presentados. A esta clase se le da el nombre de **sincronización**.

Los componentes compuestos de Dexter pueden ser usados para “recoger” un conjunto de componentes atómicos que van a ser presentados juntos. Pero esta definición no nos ofrece un mecanismo para especificar ninguna relación temporal. El problema es la sincronización entre esos componentes. Así la sincronización puede estar basada bien en información estructurada, o bien en el contenido de un componente.

Hay varias maneras de afrontar el problema de la colección y sincronización dentro de un hipermedia. A continuación mostramos tres diferentes estrategias:



(a) Estructura oculta

La manera más básica y más frecuente de manejar los datos dependientes del tiempo consiste en poner todos estos datos dentro un único componente. En el diagrama vemos que tenemos cuatro media diferentes, y queda explicitado el instante en que son activados cada uno de ellos. Esta solución no requiere apenas ningún cambio en relación al modelo de Dexter. Todo queda definido en términos de la capa del componente y la sincronización usa técnicas internas (ocultas). Como contrapartida, no permite definir combinaciones de datos más complejas.

(b) Estructura separada

Éste es el caso completamente opuesto. Aquí, cada parte de información multimedia forma un bloque diferenciado. La colección queda representada mediante un enlace multi-destino que apunte al inicio de todos los bloques (mientras que en la solución anterior únicamente era necesario un único enlace que apuntara al inicio del componente). Cruzar el enlace significará activar ese componente. Mientras que la colección es fácilmente representable, la sincronización no lo es tanto, ya que hace falta algún mecanismo para determinar cuándo un componente debe iniciarse en relación a los otros.

Por tanto, los inconvenientes de este método son que añaden una gran complejidad dado el gran número de enlaces que son necesarios crear y mantener. Cualquier herramienta tendrá que ser compleja y con una difícil interfaz de usuario.

(c) Estructura compuesta

Ésta es la solución intermedia, compromiso entre colección y sincronización. Aquí varios elementos son agrupados en componentes compuestas. Estas componentes compuestas pueden agrupar uno o varios medias.

La colección es más simple que en la segunda solución y el problema de la sincronización queda reducido a relaciones internas.

A menudo, la solución no será simplemente elegir entre uno de estos tres enfoques. De hecho esto sólo ocurrirá con hipermedias pequeños. Lo habitual será descomponer el problema en pequeños sub-problemas, a cada uno de los cuales sí que les aplicaremos una de estas tres soluciones según nos convenga.

3.1.3. Enlaces en hipermedia

Los enlaces influyen tanto en el orden de la presentación, como en los componentes que se visualizan simultáneamente, pero no son éstos sus aspectos fundamentales, ya que el objetivo de un enlace debería ser definir un mecanismo de navegación lógica en un documento.

Un enlace es aquello que define una transición de un componente a otro. Sin embargo, la noción de enlace dentro de un componente es algo que no está considerado en Dexter.

En un sistema de hipertexto, es sencillo definir un enlace, ya que un componente puede ser dividido en pequeñas partes (capítulos, secciones, ..., palabras y caracteres), y así podemos utilizar el mecanismo de anclaje. Pero en un sistema multimedia, la asociación de un enlace con o dentro de un componente es mucho más compleja, ya que los datos a menudo no pueden ser fragmentados ni indexados.

Otro problema que no se trata en Dexter es el que se refiere a cuánta información deja el usuario al navegar a través de un enlace. En un sistema de hipertexto, al cruzar un enlace, o bien se abre una nueva ventana, o bien se sustituye lo que se estaba visualizando por la información nueva. Esto es indicado aquí porque el usuario sólo puede leer una cosa a la vez. Pero en un entorno hipermedia, puede interesarle, por ejemplo, seguir oyendo una explicación, viendo el mismo vídeo, pero pasar a ver una nueva imagen o texto, dependiendo del lugar en que se encuentre de la aplicación. Para tratar esto, se introduce la noción de **contexto del enlace**, que será tratado más adelante.

3.1.4. Atributos de especificación de la presentación a alto nivel

En una aplicación hipermedia típica, cada pantalla contiene unos elementos de datos propios; pero, además, todas las pantallas tienen elementos que comparten un conjunto común de atributos para un tipo de datos en concreto. Por ejemplo, imaginemos un audio que se utiliza a lo largo de una presentación. En cada pantalla se usan diferentes conjuntos de datos de audio, pero todas las pantallas compartirán unos atributos como el volumen o la calidad del tono, los cuales no queremos que cambien cada vez que pasemos de una pantalla a otra. Por tanto, estos atributos tienen una definición global para toda la presentación y serán asociados no sólo a un componente, sino a toda un tipo de información (o medio).

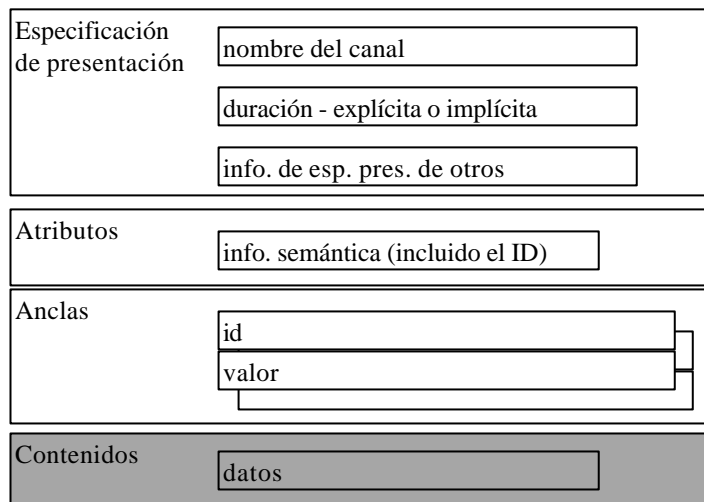
Estos atributos globales, que no se consideraron en el modelo de Dexter, se manejan utilizando el mecanismo de **canales**, que veremos más adelante.

3.2. El modelo de hipermedia de Amsterdam

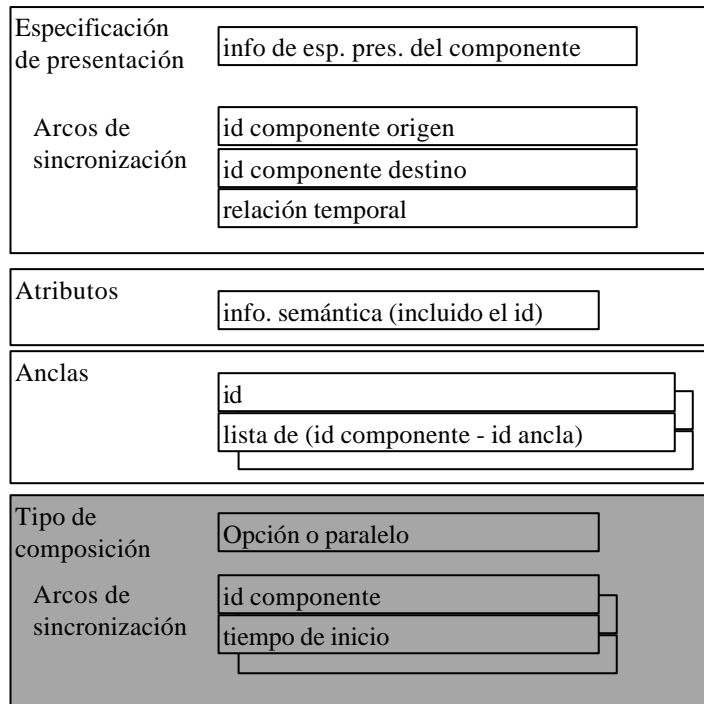
Este modelo se definió a partir de la combinación del modelo de Dexter con los canales, añadiendo extensiones para soportar ciertos requerimientos de la hipermedia que no habían sido contemplados. ha sido empleado para desarrollar numerosas aplicaciones hipermedia.

Veamos cuál es la estructura de un componente, ya sea atómico o compuesto en el modelo de Amsterdam y comparémoslo con el de Dexter, que ya vimos en el apartado anterior.

Componente atómico:



Componente compuesto:



Vemos que un componente compuesto no contiene datos, sino que sólo los datos sólo pueden ser referenciados a través de los componentes atómicos.

Este enfoque tiene tres ventajas sobre Dexter:

- Localiza la información temporal y de presentación en las componentes atómicas, mientras que la estructura de la presentación se deja para las componentes compuestas. En teoría, esto facilita las tareas de mantenimiento.
- Promueve la reusabilidad de datos al forzar que todos los elementos se mantengan por separado.
- Modela de forma más aproximada la forma en que la información multimedia se almacena, esto es, en sistemas de ficheros o bases de datos.

Así, los componentes atómicos contienen información sobre la presentación, atributos, información de anclas y campo de contenidos (datos). Lo nuevo respecto al modelo de Dexter es el campo de presentación, en el cual una se dedica a aspectos relacionados con el tiempo.

En cuanto a los componentes compuestos, se añaden varios elementos nuevos, respecto a Dexter. La diferencia principal es que ahora es usado para especificar la estructura de una presentación y no sólo para unir componentes para la navegación.

Los atributos temporales de la presentación se obtienen de los componentes atómicos, mientras que en la especificación de la presentación (componentes compuestos) aparecen **los arcos de sincronización**, unas nuevas estructuras que nos dan una información sobre el orden relativo de los componentes atómicos en la presentación, y que veremos posteriormente con más detalle.

Tanto las anclas como el campo de atributos son similares a Dexter, con una pequeña diferencia: en los componentes compuestos se añade un **tipo de composición** a la lista de parejas <id componente, id ancla>. Éste puede tomar dos valores:

- paralelo: se visualizan todos a la vez
- opción (*choice*): sólo se visualiza, como máximo, uno de los hijos

3.3. Relaciones temporales

Como ya hemos dicho, el principal mecanismo para representar las relaciones temporales entre entidades son las componentes compuestas.

Hay dos tipos de sincronización entre componentes, una calificada como de grano grueso y otra de grano fino. La primera es aquella que consiste en restricciones definidas entre los hijos de una componente compuesta (que recordemos que pueden ser componentes atómicas o compuestas), como por ejemplo el instante relativo de comienzo de cada hijo dentro de la componente compuesta. Esta información está explícita en la definición de cada hijo. Por contra la sincronización de grano grueso consiste en relaciones entre dos hijos (que pueden o no estar al mismo nivel, es decir que pueden o no ser hermanos) explicitados mediante lo que llamamos un arco de sincronización. Un arco de sincronización no se usa como un enlace navegacional, sino que constituye un mecanismo flexible para establecer relaciones temporales dentro del sistema hipermedia.

El modo de especificar las relaciones temporales en AHM tiene la ventaja de que hay una clara separación entre la colección de componentes que se muestran juntas, el orden relativo en el lo hacen y las relaciones temporales específicas que existen durante una presentación.

3.4. Otras contribuciones del modelo

3.4.1. Contexto de los enlaces

Los componentes compuestos no nos dan ninguna información de cómo se comporta cada componente al navegar a través de un enlace.

Para ello definimos el concepto de contexto de enlace, que será un nuevo componente (típicamente compuesto) que contiene la colección de los componentes afectados por un enlace. El mecanismo de contexto permite definir unas opciones de visualización diferentes para cada enlace. En particular, puede ser que parte de la información continúe visible, mientras que otra deja de estarlo. La ventaja de esto es que, por un lado, sólo una parte de la estructura del documento se ve afectada al seguir un enlace, y por otro, se nos ofrece una gran flexibilidad.

3.4.2. Canales

Terminaremos este apartado con el concepto de canales. Éstos son salidas abstractas utilizadas para reproducir los contenidos de un componente. Asociado con cada canal hay unas características de la presentación. Éstas pueden ser dependientes de los tipos de datos (media), como la fuente y estilo de un texto o el volumen de un sonido; pero también pueden ser independientes, como por ejemplo los colores de *background* y *foreground*.

Un uso típico puede verse en una situación en que deseamos una aplicación hipermedia en dos idiomas, por ejemplo castellano y catalán. Imaginemos que tenemos un vídeo acompañado de un audio y un texto explicativos. Necesitaríamos un canal de vídeo (el mismo si estamos en catalán o castellano), y luego dos canales de audio y otros dos de texto. Así se consigue una independencia entre los datos y la presentación, así como un gran dinamismo en estas últimas.

3.5. Conclusiones

Como ya hemos explicado el principal interés del modelo de Amsterdam estriba en el intento de extender el modelo de Dexter para que contemple la complejidad de los tipos multimedia, y muy en especial del problema del tiempo que éstos conllevan.

4. Conclusiones finales

Dexter puede ser empleado para modelar la web considerada como un hipertexto, sin tener en cuenta los servicios telemáticos como mail, chats,... Pero la separación que establece entre las capas se ve mucho más claramente aún en la arquitectura XML/XSL. En este caso hay una clara distinción entre lo que son los datos (un documento XML) que puede tener muchas presentaciones (plantillas XSL).

Respecto a las características temporales, es evidente que la web de hoy en día aún no las incorpora. Por ejemplo no podemos crear una página que nos presente algo como: "muéstrame este vídeo y a los 10 segundos reproduce este sonido y muestra este texto. Y al acabar el vídeo, muestra esta imagen, etc..." No obstante se está trabajando en una especificación para poder disponer de sincronización en la web: se trata de SMIL (Synchronized Multimedia Integration Language), un lenguaje basado en XML que permite, muy brevemente, definir secciones dentro de una presentación y sincronizar los medios que se reproducen en dichas secciones. SMIL es una recomendación del W3C y ha sido utilizado sobretodo por RealNetworks que lo introdujo en su RealPlayer G2, y también Apple, que ya lo incluye en Quicktime 4.1 (también Microsoft, Macromedia y Compaq están trabajando en HTML+time, que viene a ser una segunda versión de SMIL). En SMIL también se aprecian claramente conceptos enunciados en el modelo de Amsterdam, como las colecciones, sincronización, canales,...

5. Referencias bibliográficas

- [1] Navarrete Terrasa, Antonio: **Una metodología relacional hipermedia. Estudio en casos prácticos.** *Proyecto final de carrera de la Ingeniería en Informática, Universitat de les Illes Balears*, 1998.

- [2] F. Halasz, M. Schwartz: **The Dexter Hypertext reference model.** *Communications of the ACM*, vol. 37, pp. 30-39, 1994.

- [3] L. Hardman, D.C.A. Bulterman, G. Van Rossum: **The Amsterdam Hypermedia Model.** *Communications of the ACM*, vol. 37, pp. 50-62, 1994.