UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

Departamento de Sistemas Informáticos y Programación

CONCEPTUALIZACIÓN, PROTOTIPADO Y PROCESO DE APLICACIONES HIPERMEDIA

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Antonio Navarro Martín

Bajo la dirección de los doctores

Alfredo Fernández - Valmayor Crespo
Baltasar Fernández Manjón

Madrid, 2002

ISBN: 84-669-1802-7

Conceptualización, prototipado y proceso de aplicaciones hipermedia

Memoria que presenta para optar al grado de
Doctor en Ciencias Matemáticas

Antonio Navarro Martín

Dirigida por los profesores

Alfredo Fernández-Valmayor Crespo

Baltasar Fernández Manjón

Departamento de Sistemas Informáticos y Programación
Facultad de Ciencias Matemáticas
Universidad Complutense de Madrid

Abril 2002

Índice

1 Introducción

Ha pasado más de medio siglo desde que Vannevar Bush en el año 1945 describiese Memex [Bush 45], el protosistema referenciado por multitud de autores centrados en el dominio hipermedia. Hoy en día el desarrollo tecnológico informático ha sufrido un avance tan espectacular que ni siquiera un visionario como Bush hubiera podido imaginar. Desde mediados de los 80 los ordenadores han invadido el planeta en todos los ámbitos posibles: industriales, educativos, personales, etc. En un primer momento esta invasión pacífica estuvo marcada por el confinamiento del ordenador a nivel personal, o a lo sumo, a una red interna. Pero a mediados de los 90 esta situación cambió debido a la interconexión de estos invasores a través de una telaraña que se extiende a nivel mundial, Internet, y en particular la *World Wide Web*. La Web, no es más que una vasta cantidad de información interconectada básicamente a través de hiperenlaces HTML. Cincuenta años después el sistema descrito por Bush se hizo realidad, aunque en este caso la información a la que tiene acceso el usuario no se restringe a la introducida por el mismo, sino a aquella que pueda obtener desde su sistema, de la cual la mayor parte no habrá sido introducida por él. Por tanto, desde este punto de vista, la Web puede verse como una gigantesca aplicación hipermedia formada por multitud de información heterogénea y distribuida.

La exitosa evolución de la Web ha eclipsado parcialmente sistemas hipermedia cruciales en el desarrollo de esta área de la informática, tales como Augment, Xanadu, KMS, NoteCards, Intermedia, o HyperCard [Conklin 87], [Nielsen 95]. Hoy en día estos sistemas se están viendo reemplazados por multitud de herramientas para el desarrollo de aplicaciones hipermedia, y en particular de aplicaciones Web [Fraternali 99]. De los sistemas propietarios que servían para construir aplicaciones hipermedia hemos pasado a aproximaciones mucho más genéricas, con diseños basados en patrones software reutilizables [Nanard 99], [Rossi 00], desarrollos fundamentados en el uso de componentes software [Will 99], sistemas abiertos [Davis 92], o aplicaciones con avanzadas capacidades de sincronización [W3C SMIL]. En base a la naturaleza heterogénea de las aplicaciones generadas por estas aproximaciones cabe preguntarse ¿qué tienen en común todas estas aplicaciones para considerarlas como evoluciones del sistema Memex propuesto por Bush hace más de cincuenta años? Dicho de otra forma, ¿qué tienen en común estas aplicaciones para ser consideradas aplicaciones hipermedia? La respuesta es bastante sencilla, están formadas por información de diversa naturaleza (texto, audio, vídeo, etc.), la cual se encuentra organizada mediante una red hipertextual. Los primeros métodos de modelado específicos del área [Campbell 88] fueron conscientes de que la naturaleza hipertextual de estas aplicaciones se correspondía básicamente con un grafo dirigido. El problema que surge de esta aproximación es que la navegación que el usuario lleva a cabo por el grafo, no tiene porque corresponderse con la estructura del mismo, pudiendo visitar de manera simultánea más de un nodo. Para paliar este problema surgieron extensiones de la noción de grafo [Tompa 89], [Stotts 89] con el fin de capturar la semántica de navegación de la aplicación hipermedia, es decir, el modo en que la información va a ser visitada y presentada al usuario [Stotts 89].

Por otro lado, las aplicaciones hipermedia son aplicaciones software, y como tal necesitan un modelo de proceso bien definido que guíe su desarrollo. En particular, a nosotros nos parece especialmente acertado el propuesto por Fraternali [Fraternali 99] que divide la construcción de este tipo de aplicaciones en un primer bucle de conceptualización-prototipado, y otro de diseño-construcción. Si bien la fase de diseño debe estar íntimamente ligada a las tecnologías de desarrollo, y la naturaleza de la aplicación (orientada a objetos, sistemas abiertos, con fuerte componente síncrona, etc.), la fase de conceptualización es, y debe ser, ajena a técnicas concretas de desarrollo, ya que en esta fase solamente interesa identificar los contenidos de la aplicación, sus relaciones hipertextuales y la forma de presentar los mismos al usuario. Parece razonable por tanto proporcionar un mecanismo específico que sirva para cubrir las fases de conceptualización y prototipado en el modelo de proceso de desarrollo de aplicaciones hipermedia que no comprometa las fases de diseño y construcción. Aunque los sistemas de representación hipermedia (modelos, metodologías, etc.) clásicos son básicamente herramientas de diseño [Nanard 95], una primera aproximación podría intentar adaptarlos para cubrir la fase de conceptualización [Fraternali 99]. Características deseables para estos sistemas serían que fuesen capaces de representar la información proporcionada por el modelo Dexter, incluyendo además capacidades de sincronización y de representación de contextos del modelo Amsterdam [Hardman 93]. También sería deseable que fuesen capaces de modelar contenidos generados dinámicamente, característica clave en las últimas aplicaciones desarrolladas para la Web [Bodner 00], [Göschka 99]. Finalmente sería deseable contar con semántica de navegación por defecto, de tal forma que al imponerla mediante un mecanismo genérico que no necesitase de ningún código o algoritmo proporcionado por el desarrollador, sirviese para describir la manera en que se va a presentar la información al usuario. Esta característica es clave para la generación automática de prototipos, los cuales serán evaluados en la fase posterior a la conceptualización [Nanard 98]. Pues bien, los sistemas de representación hipermedia de mayor

influencia en el campo [Halasz 94], [Hardman 93], [Campbell 88], [Tompa 89], [Stotts 89], [Garzotto 93], [Isakowitz 95], [Díaz 94], [Schwabe 96] y [Millard 00] no tienen estas tres características simultáneamente [Navarro 02]. Además estudios recientes [Barry 01] demuestran que estos sistemas de representación no se utilizan con profusión en el mundo empresarial, prefiriéndose técnicas genéricas de diseño y programación que no limiten el desarrollo de aplicaciones hipermedia, tal y como hacen los formalismos anteriores. El problema radica en que los sistemas de representación hipermedia clásicos están concebido como herramientas para obtener representaciones de diseño en vez de ayudar al proceso de diseño.

En esta tesis doctoral se proporciona un modelo hipermedia, el modelo *Pipe* [Navarro 02], que presenta las tres características anteriores, siendo por tanto un mecanismo que cumple los requisitos mínimos para cubrir las fases de conceptualización y prototipado. Además, al estar centrado en conceptualización y no en diseño, está concebido como una ayuda o complemento a este, permitiendo utilizar cualquier técnica concreta de implementación. El objetivo final del desarrollo de este modelo hipermedia es introducirlo en el modelo de proceso propuesto por [Fraternali 99] (e influenciado por [Ginige 97]) obteniendo de esta forma un modelo de proceso dirigido por Pipe, al cual denominaremos *Plumbing* [Navarro 02]. Este modelo de proceso compromete únicamente el uso de Pipe para dirigir la fase de conceptualización y prototipado, permitiendo la incorporación de otros sistemas de representación, y de técnicas específicas en las fases de diseño y construcción.

Especializando Plumbing con técnicas concretas de representación de las estructuras Pipe (XML), y con un lenguaje capaz de soportar la semántica de presentación por defecto del mismo (Java) seremos capaces de construir de maneara automática aplicaciones, facilitando de esta forma la fase de prototipado. Al modelo de proceso resultante de esta especialización lo denominaremos *PlumbingXJ* [Navarro 02]. La idea es utilizar la potencia expresiva de los lenguajes de marcado para proporcionar una descripción del modelo Pipe más cercana al usuario (de manera similar a lo que sucede con el modelo Amsterdam y el lenguaje SMIL). Al utilizar esta descripción junto con diversas tecnologías XML seremos capaces de generar prototipos de manera automática. Nótese que esta aproximación, no excluye reutilizar totalmente las técnicas de conceptualización y prototipado en diseño y desarrollo, obteniendo de esta forma una metodología concreta de desarrollo de aplicaciones hipermedia. Este punto queda como trabajo futuro.

En base a lo anterior, esta tesis está organizada en diversos capítulos y apéndices. Los primeros capítulos se dedican a presentar el estado del arte. A continuación presentamos los principales resultados de nuestra investigación. Finalmente y tras exponer las conclusiones y el trabajo futuro, se incluyen una serie de apéndices que complementan la información de los anteriores capítulos. Veamos los contenidos con un poco más de detalle.

El Capítulo 2 presenta un recorrido y análisis crítico de los más notables sistemas de representación utilizados en el diseño de aplicaciones hipermedia. El esquema general de estudio de cada sistema consiste en una presentación previa del mismo. A continuación se utilizan las capacidades expresivas de cada sistema de representación para caracterizar un ejemplo de aplicación. Mantenemos en todos los sistemas el mismo ejemplo para un mejor estudio comparativo entre los mismos. Finalmente procedemos a una comparación del sistema estudiado con los demás.

El Capítulo 3 estudia las principales técnicas de desarrollo de aplicaciones hipermedia. Debido al gran número de estas hemos optado por restringirnos a tres grandes aproximaciones. La primera viene representada por los modelos de proceso específicos para el dominio hipermedia. La segunda se deriva de las metodologías concretas asociadas a los sistemas de representación hipermedia comentados en el Capítulo 2. La tercera y última, es un estudio de una metodología concreta para el desarrollo de aplicaciones software genéricas, DTC, creada en el seno de nuestro grupo de investigación, y que ha sido aplicada con éxito en el área hipermedia.

El Capítulo 4 completa el estado del arte haciendo un estudio y análisis del área de los lenguajes de marcado, y de sus más notables aportaciones al dominio hipermedia. Así, empezamos estudiando los principales formalismos para la definición de lenguajes de marcado. Después nos centramos en los desarrollos de lenguajes de marcado específicos para hipermedia, y para representación de interfaces de usuario. Finalmente hacemos mención a otros estándares relacionados que se utilizarán en el desarrollo técnico de nuestra investigación.

El Capítulo 5 presenta a Pipe, el modelo hipermedia que hemos desarrollado con el fin de utilizarlo como herramienta de conceptualización. Como ya hemos comentado, este modelo se ajusta a los preceptos del

modelo Dexter, manteniendo las nociones de contexto y sincronización del modelo Amterdam. También permite caracterizar aplicaciones dinámicas, y cuenta con una semántica de presentación por defecto. En este capítulo también incluimos un ejemplo que muestra las capacidades expresivas de nuestro sistema de representación hipermedia, y un análisis comparativo con los sistemas estudiados en el Capítulo 2.

El Capítulo 6 muestra el resultado de utilizar a Pipe como herramienta de conceptualización en el modelo de proceso de Fraternali, obteniendo el modelo de proceso Plumbing. Este modelo de proceso se beneficia de las características de Pipe para resolver los principales inconvenientes de aplicabilidad de los sistemas de representación hipermedia en el desarrollo a gran escala. Este capítulo también describe a PlumbingXJ, el resultado de aplicar lenguajes de marcado para representar a las estructuras Pipe, y de técnicas de programación orientadas a objetos para construir un sistema hipermedia basado en dichas descripciones.

Pensamos que un área como la ingeniería del software necesita desarrollos reales para contrastar la viabilidad de las técnicas propuestas. Con este fin, en el Capítulo 7 aplicamos PlumbingXJ a cuatro aplicaciones hipermedia ya desarrolladas haciendo un ejercicio de ingeniería inversa. Tres de ellas fueron desarrolladas en el ámbito de nuestro grupo de investigación utilizando técnicas afines a las enunciadas por PlumbingXJ. La última es el sitio web de Amazon, rico en contenidos y enlaces generados dinámicamente.

El Capítulo 8 concluye la investigación presentada en esta tesis, resumiendo las principales conclusiones, y enumerando las líneas de trabajo e investigación, presentes y futuras, que se derivan de ellas.

El Capítulo 9 recoge las referencias en las que se ha basado tanto nuestro estudio del estado del arte, como la línea de investigación presentada. El Apéndice A muestra una extensión de Pipe capaz de caracterizar aplicaciones con enlaces n-arios. No podríamos escribir una tesis en el área de la ingeniería del software sin prestar una especial atención al diseño de las herramientas software desarrolladas en su seno. Por tanto el Apéndice B muestra el diseño del sistema hipermedia utilizado en PlumbingXJ. El Apéndice C recoge las representaciones documentales Pipe de las diversas aplicaciones hipermedia que hemos generado durante la aplicación de PlumbingXJ a lo largo de esta tesis. Finalmente el Apéndice D da cabida a toda la información que hemos considerado relevante en la creación de los capítulos de esta tesis, pero cuya naturaleza la hacia acreedora de aparecer en un apéndice.

2 Sistemas de representación hipermedia

2.1 Introducción y clasificaciones

En este capítulo analizaremos diversos *Sistemas de Representación Hipermedia*, SRH, de distinta naturaleza. La razón se debe a que uno de los principales resultados de este trabajo es el sistema de representación hipermedia propuesto en el Capítulo 5. Por lo tanto haremos un repaso de los más relevantes para luego poder compararlos con el de propia creación. Pero ¿a qué denominamos sistema de representación hipermedia?, pues simplemente a cualquier formalismo que permita representar una aplicación hipermedia de manera abstracta. A nuestro entender, básicamente hay tres tipos de sistemas de representación hipermedia: (i) *Modelos de referencia*, que permiten describir a los sistemas que crean y gestionan aplicaciones hipermedia, es decir a los *sistemas hipermedia*; (ii) *Modelos hipermedia*, que son representaciones abstractas de aplicaciones hipermedia; y (iii) *Metodologías de diseño*, que son un conjunto de técnicas para diseñar aplicaciones hipermedia. Como podemos comprobar en base a las definiciones anteriores, los modelos hipermedia son los que mejor encajan en la definición de sistema de representación hipermedia, y por tanto, son los de un uso más directo a la hora de caracterizar aplicaciones de este tipo. A pesar de este hecho, tanto los modelos de referencia, como las metodologías también pueden utilizarse para caracterizar aplicaciones hipermedia. En efecto, los modelos de referencia sirven para describir los sistemas hipermedia con los que se crean aplicaciones hipermedia. En la descripción de estos sistemas se utilizan representaciones de las aplicaciones hipermedia que los sistemas deben ser capaces de construir. Precisamente esta es la característica que los capacita como sistemas de representación hipermedia. Las metodologías son una serie de pasos y notaciones para representar el diseño de aplicaciones hipermedia. A partir de ese diseño se puede construir una aplicación. Por lo tanto podemos tomar el diseño como método de representación de la aplicación.

Según nuestro criterio, si agrupamos los sistemas de representación hipermedia que vamos a estudiar en este capítulo en base a la clasificación anterior obtenemos lo siguiente. Modelos de referencia son la *Máquina Abstracta de Hipertexto*, HAM [Campbell 88], y el *Modelo Dexter* [Halasz 94]. Modelos hipermedia son el *Modelo Ámsterdam* [Hardman 93], el *Modelo de Hipergrafos* [Tompa 89], el *Modelo Trellis* [Sttots 89], el *Modelo de Diseño Hipermedia*, HDM [Garzotto 93], el *Modelo de Datos de Gestión de Relaciones*, RMDM [Isakowitz 95], y el *Modelo Labyrinth* [Díaz 97]. Finalmente entre las metodologías tenemos, el *Modelo de Gestión de Relaciones*, RMM [Isakowitz 95] (que utiliza el modelo hipermedia RMDM), y el *Modelo de Diseño Hipermedia Orientado a Objetos*, OOHDM [Schwabe 95a].

En el estudio proporcionado en este capítulo no vamos a seguir la clasificación anterior, ya que vamos a primar, las que a nuestro entender son las características expresivas fundamentales de los sistemas de representación hipermedia, en vez de su naturaleza. Además, si un modelo de referencia al caracterizar un sistema hipermedia puede caracterizar las aplicaciones que ejecuta, un modelo hipermedia al caracterizar muchas aplicaciones, en la práctica también puede usarse para caracterizar al propio sistema hipermedia. Cómo podemos comprobar, esta diferencia es extremadamente sutil, e incluso dependiente de la terminología utilizada por los autores.

Por lo tanto, si consideramos las principales aportaciones de los distintos sistemas de representación hipermedia tenemos que las del modelo Dexter son: *a)* la separación de contenidos, almacenamiento y presentación. El modelo Amsterdam aporta *b)* la noción de contexto, y *c)* sincronización. Los SRH HAM, hipergrafos y Trellis aportan *d)* formalización explícita de la semántica de navegación de la aplicación (es decir, que contenidos se muestran al usuario cuando este navega por el hipertexto). HDM y RMDM aportan *e)* una organización relacional de los contenidos. Labyrinth aporta una *f)* gestión explícita de eventos, *g)* la posibilidad de caracterizar contenidos y enlaces generados dinámicamente, y *h)* gestión de usuarios y seguridad. Finalmente OOHDM aporta una *i)* distinción entre los enlaces a nivel de contenidos y los enlaces a nivel de navegación, y *j)* la inclusión de técnicas orientadas a objetos. En la enumeración anterior no se excluye que algunos SRH tengan características ya expuestas por otros sistemas. Así, por ejemplo, el modelo Labyrinth tiene soporte de las nociones de contexto y sincronización Ámsterdam. Este hecho no tiene importancia en la práctica, ya que en la clasificación hecha en este capítulo no se tiene en cuenta que sistema aportó que característica, sino cual es la característica más distintiva de cada sistema.

En base a la clasificación anterior agruparemos los sistemas de referencia hipermedia de la siguiente forma. Los que sirven como modelo de referencia con tres niveles de separación (Dexter), los que están basados en fuertes restricciones de sincronización (Amsterdam), los que presentan una formalización de la semántica de

navegación de la aplicación (HAM, hipergrafos, y Trellis), los relacionales (HDM y RMDM), y finalmente aquellos que son capaces de responder a eventos genéricos (Labyrinth y OOHDM). Esta clasificación no es la única posible, y puede ser incluso discutible, pero desde nuestro punto de vista es la más natural.

Aunque existen otros sistemas de referencia hipermedia bastante interesantes, no han sido analizados debido a su baja repercusión en el contexto hipermedia [Paulo 99], [Muchaluat 01], o por que su relativa nueva incorporación no ha permitido un uso lo suficientemente amplio como para incluirlo en este capítulo [Millard 00]. Finalmente la propuesta de [Mandel 98] y [Baumeister 99] no la hemos incluido por ser muy similar a OOHDM.

Como podemos ver existen multitud de SRH y de naturaleza muy heterogénea. La razón de este gran número de modelos (y SRH por extensión) la explica [Garzotto 93], enumerando las ventajas que se derivan de la utilización de estos. Los modelos hipermedia permiten una *mejora de la comunicación*. Un modelo de diseño proporciona un lenguaje que puede ser utilizado por los desarrolladores para especificar la aplicación, y para guiar la construcción sirviendo como plataforma de comunicación. Sirven para *desarrollar metodologías de diseño y estilos retóricos*. Estos modelos proporcionan un marco en el que los autores de aplicaciones pueden desarrollar, analizar y comparar metodologías y estilos retóricos de autoría, con un alto nivel de abstracción. Mejoran la *reusabilidad*. El disponer de un lenguaje de modelado permite reutilizar la parte central de la estructura de las aplicaciones, ya que las especificaciones basadas en un modelo capturan la semántica esencial de la aplicación, y por tanto, pueden ser reutilizadas cuando las semánticas de las aplicaciones sean lo suficientemente similares. También *proporcionan entornos de lectura consistentes y predecibles*. Las herramientas para especificar estructuras de hipertexto ayudan a los autores a prevenir inconsistencias estructurales y errores, y además, las aplicaciones desarrolladas según un modelo tienen estructuras de representación muy consistentes y predecibles. Como consecuencia, los entornos de navegación también serán predecibles, ayudando a eliminar los problemas de desorientación del lector. Finalmente los modelos pueden ser utilizados por *herramientas de diseño*. Estos modelos son la base para el desarrollo de herramientas de diseño que den soporte a un proceso de desarrollo sistemático y estructurado, permitiendo que el diseñador trabaje a un nivel de abstracción más cercano al dominio de la aplicación, y proporcionando un proceso de traducción sistemático al nivel de implementación.

Antes de empezar el estudio haremos una pequeña digresión sobre el término hipermedia e hipertexto. En principio, se puede considerar una aplicación hipermedia como un hipertexto con contenidos ampliados. De esta forma, si antes aparecía texto como contenido válido, ahora puede aparecer una imagen, o un vídeo. Desde este punto de vista, las aplicaciones hipermedia e hipertextuales son casi equivalentes. En la medida que aparezcan sutiles relaciones de presentación espacial y de sincronización, la aproximación anterior empieza a fallar, necesitándose sistemas de representación con mayor potencia a la hora de representar estas relaciones (como Amsterdam, Labyrinth, u OOHDM). De todas formas la aproximación anterior tampoco carece de fundamento. La World Wide Web, que puede ser considerada como la mayor aplicación hipermedia del mundo, se basa en HTML, lenguaje que no dispone de ninguna capacidad de sincronización. En la actualidad se intenta paliar esta deficiencia con lenguajes más orientados al mundo hipermedia como SMIL [SMIL], pero hasta ahora su éxito es limitado. Por lo tanto en la exposición siguiente utilizaremos indistintamente el término hipertexto y el término hipermedia, respetando el término original del modelo en cuestión.

La forma de presentar los sistemas de referencia hipermedia (limitados, claro está, por el material disponible de cada uno de ellos) será la siguiente: primero se agruparan en base a la característica más relevante, después se presentará una descripción del mismo, se verá un ejemplo de aplicación (común a todos) y finalmente se contextualizará y discutirá con el resto de sistemas de referencia.

2.1.1 Ejemplo

A lo largo de este capítulo utilizaremos un ejemplo común para ilustrar como caracterizan los diversos sistemas de representación hipermedia los aspectos más comunes de las aplicaciones hipermedia.

Supongamos que queremos modelar los contenidos de la página Web de un profesor de universidad (para centrar ideas elegiremos al autor de estos párrafos). En dichos contenidos aparecerá un texto inicial que contendrá una breve biografía del profesor. Esta biografía enlaza con un mapa España, país de nacimiento del autor, y viceversa (es decir, el mapa de España también presenta un enlace con la biografía). Dentro de este

mapa tenemos enlaces con descripciones de diversas ciudades de la geografía española que son de interés para el profesor, y viceversa. En particular destacan las ciudades de Madrid, y la localidad de Lepe. La biografía también presenta un enlace a una descripción los estudios cursados por el profesor, y viceversa. Finalmente la biografía enlaza con una descripción de la docencia que imparte el profesor y viceversa. Además, la biografía permite enlazar con descripciones concretas a cada asignatura dentro de la descripción genérica. La descripción de asignaturas enlaza con el programa de cada una de ellas y viceversa. Los programas enlazan con su biografía correspondiente.

También disponemos de un formulario de búsqueda, de tal forma que cuando el usuario introduzca un término se generará un contenido con enlaces a los resultados de la búsqueda. La Figura 2.1 muestra un grafo que describe los contenidos y sus relaciones a través de los enlaces.

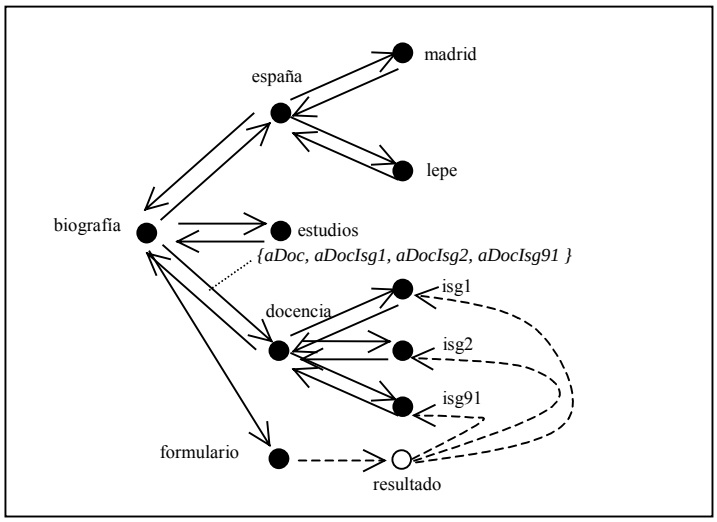

Figura 2.1 Grafo que ilustra los contenidos de la aplicación y su enlaces. Los círculos en blanco representan contenidos dinámicamente generados, y las flechas discontinuas los enlaces también generados dinámicamente. Nótese que los componentes dinámicos pueden variar en cada ejecución, y los aquí representados son un ejemplo.

Además queremos proporcionarle el siguiente esquema navegacional: en una primera ventana aparecerá la biografía. Cuando se seleccionen los enlaces de la biografía se accederá a una segunda ventana con dos paneles. En uno se mostrarán los enlaces provenientes de la primera ventana. En el otro, los enlaces provenientes del primer panel de esta segunda ventana. Además, en este último panel se mostrarán el resto de posibles contenidos. Como sabemos que la notación para referirse a interfaces de usuario no es homogénea, la Tabla 5.4 del Capítulo 5 ilustra la notación utilizada en esta tesis, la cual coincide con la entrada de la tabla correspondiente a PlumbingXJ (descrito en el Capítulo 6). La Figura 2.2 (generada con la herramienta de prototipado descrita en el Capítulo 6) muestra la interfaz descrita en este párrafo.

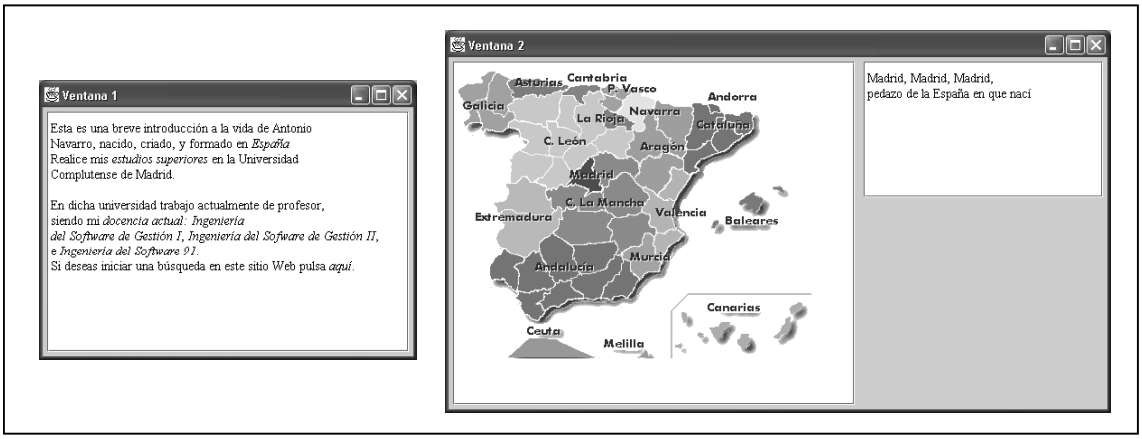

Figura 2.2 Ventanas de la aplicación. La Ventana 1 muestra la biografía. La Ventana 2 muestra españa y madrid.

2.2 Modelos de referencia

Dentro de este apartado solo aparece el modelo Dexter. La razón se debe a que al describir sistemas hipermedia y no aplicaciones, el modelo Dexter es el más genérico de todos, pudiéndose considerar la mayoría de modelos como concreciones más o menos completas del mismo.

2.2.1 El modelo Dexter

2.2.1.1 Presentación

Introducción

El modelo Dexter [Halasz 94] surge originalmente en 1990 como un intento de formalización y abstracción de los sistemas hipermedia existentes hasta ese momento. Por tanto no es tanto un modelo para representar una aplicación hipermedia concreta, como un modelo para representar a los sistemas con los que se crean tales aplicaciones, es decir un modelo de referencia. Lo que sucede es que por extensión, al tratar de modelar a los sistemas, el modelo es capaz de representar a las aplicaciones hipermedia creadas con tales sistemas.

El modelo divide a un sistema hipertexto (y por tanto a la propia aplicación hipertextual) en tres capas o niveles: el *nivel de composición interna* (o nivel interno según [Díaz 97]), el *nivel de almacenamiento,* y el *nivel de ejecución* (o nivel de presentación según [Díaz 97]), como ilustra la Figura 2.3. El primer nivel considera a los contenidos que van a aparecer en la aplicación hipermedia (nótese que esta ocasión si utilizamos la palabra hipermedia en lugar de hipertexto). El segundo considera una red de nodos donde aparecen los contenidos, y enlaces entre ellos. Finalmente el nivel de ejecución presenta el hipertexto a los usuarios para su visualización o para su modificación. Evidentemente estos niveles están íntimamente relacionados unos con otros. La forma de relacionarlos es mediante las anclas, en el primer caso, y mediante especificaciones de presentación en el segundo. Así los nodos del nivel de almacenamiento están formados por componentes, que actúan como contenedores de datos, estando en estos componentes incluidas las anclas (origen y destino de los hiperenlaces). Por otro lado las especificaciones de presentación son un mecanismo que indican como se va a presentar al usuario los componentes y enlaces del nivel anterior. Veamos el modelo con un poco más de detalle.

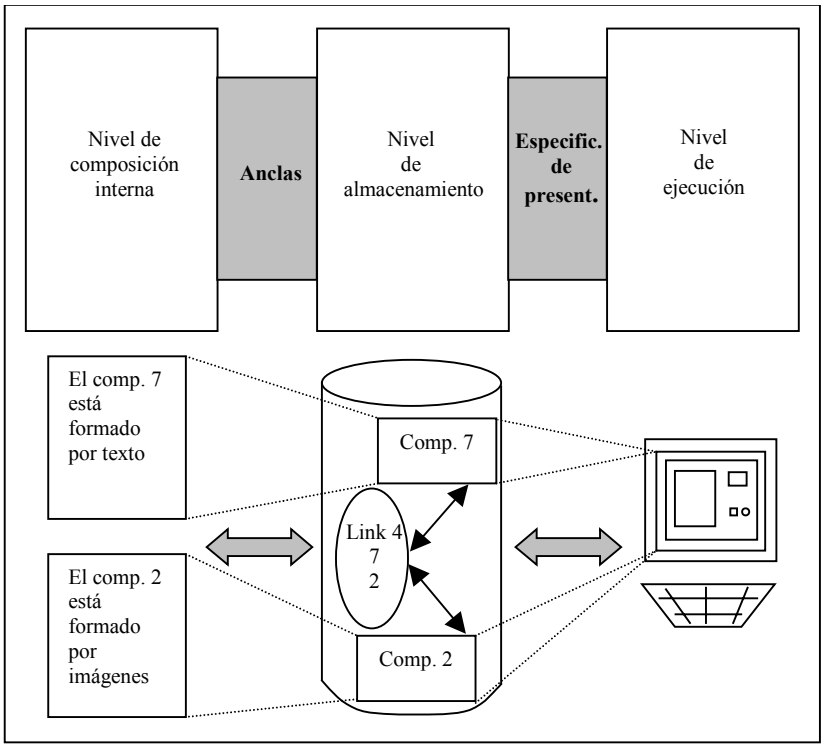

Figura 2.3 Los tres niveles del modelo Dexter

El nivel de composición interna

Este nivel se centra en los contenidos que pueden aparecer en la aplicación hipermedia, y en la estructura interna de los mismos. Tiene un carácter meramente declarativo, ya que el modelo no se preocupa de utilizar representaciones de los contenidos. La razón es evidente, ya que en principio no se restringe el rango de los posibles contenidos que pueden aparecer en una aplicación: texto, gráficos, animaciones, simulaciones, imágenes, etc. (nótese como el modelo, aún denominándose de hipertexto también considera la aparición de datos multimedia dentro de la aplicación). Por estos motivos los creadores consideran que no es realista intentar modelar todos los tipos de datos posibles. En su lugar, el modelo Dexter considera al nivel de composición interna como algo externo al mismo, y supone que existirán otros modelos de referencia diseñados específicamente para representar la estructura de aplicaciones, documentos, tipos de datos concretos (SGML, por ejemplo). Esta misma aproximación será la elegida por nuestro modelo que presentaremos en el Capítulo 5.

El nivel de almacenamiento y las anclas

Este nivel es el central en el modelo, y como ya hemos comentado se encarga de representar la estructura de nodos y enlaces del hipertexto. Los autores del modelo decidieron explícitamente evitar el término nodo debido a la controversia que suscitaba (El modelo Dexter fue el resultado de dos *workshops* de hipertexto, e intentaba capturar y formalizar los sistemas hipertexto allí presentados). En su lugar utilizaron el término componente. Un componente puede ser un átomo, un enlace, o una entidad compuesta formada por otros componentes. Los componentes atómicos están formados por los elementos básicos del nivel de composición interna, es decir, texto, imágenes, audio, etc. Los elementos compuestos están formados por una serie de componentes, de tal forma que su relación sea de grafo dirigido acíclico. Los enlaces son entidades que representan relaciones entre otros componentes. Básicamente son una secuencia de dos o más especificaciones de puntos finales (del inglés *end-point*), cada una de las cuales se refiere a un componente del hipertexto. Cada componente tiene un identificador único denominado UID (*Unique IDentifier*), que se supone único en todo el universo del discurso.

En este nivel también se disponen de dos funciones, la *función de resolución*, y la *función de acceso*.

$$\text{resolución}: \text{Especificaciones} \rightarrow \text{UIDs}$$
$$\text{acceso}: \text{UIDs} \rightarrow \text{Componentes}$$

Estas funciones se utilizan de forma conjunta para recuperar componentes, es decir, traducir especificaciones de componentes en los propios componentes. Si solamente se desea utilizar referencias a componentes basadas en el UID, se proporciona dicho identificador a la función de acceso, la cual devuelve el componente referenciado por el identificador (nótese que esta función es similar a la función `id()` de XPath). Si por el contrario se permiten referencias a componentes mediante especificaciones del tipo "el componente que contiene la palabra Prototipos", debemos saber antes que componente es ese (y si existe). Con este fin se provee la función de resolución, la cual es capaz de resolver especificaciones genéricas en UIDs concretos. Esta función es parcial, ya que es posible que algunas especificaciones no sean trasladables a un UID concreto. Nótese que, por lo general, mecanismos de localización basados en la función de resolución pueden resultar inaceptablemente costosos, sobre todo en entornos distribuidos. Por tanto parece más realista utilizar directamente los UIDs de los componentes, siendo en este caso la función de resolución la identidad.

Hasta ahora hemos hablado de enlazar componentes con componentes, pero en las aplicaciones hipermedia reales es posible especificar enlaces de partes de componentes a partes de componentes. La forma de localizar estas partes de los componentes dentro de los mismos se resuelve a través del concepto de *ancla*. Un ancla tiene dos partes: un *valor de ancla*, y un *identificador de ancla*. El valor de ancla es un valor arbitrario que especifica alguna localización, región, elemento, o subestructura dentro del componente. Por ejemplo el valor `(234, 7)` puede servir para identificar los caracteres `234` a `241` de un texto. Este valor de ancla es interpretable únicamente por las aplicaciones responsables de manejar el contenido del componente, y no está restringido dentro del nivel de almacenamiento. La razón es bastante evidente, ya que si el nivel de composición interna no modelaba la estructura interna de los datos que aparecen en los componentes, ahora desde el nivel de almacenamiento no podemos referenciar de forma abstracta el contenido de dichos componentes. Al igual que con el nivel de composición interna, el modelo hipermedia presentado en el Capítulo 5 sigue esta misma aproximación con las anclas. El identificador de ancla es un identificador único en el entorno del componente. De esta forma las anclas pueden ser identificadas de forma única por la pareja formada por el UID del componente y el identificador del ancla. Esta aproximación es equivalente a

anteponer el valor del UID delante de cada valor de ancla (por supuesto con un carácter separador) y a continuación dar el valor de identificador de ancla. La consideración del valor del ancla y del identificador de ancla de forma separada permite que el contenido al que está adosada el ancla varíe, con el valor del ancla, sin modificar el identificador de la misma.

Las anclas sirven para denotar el origen y/o destino de los enlaces dentro del nivel de almacenamiento. De esta forma los enlaces (no olvidemos que son componentes, y por tanto entidades del mismo nivel que por ejemplo una foto) contienen *especificadores* (del inglés *specifier*) formados por una especificación de componente (en particular su UID), una *dirección* de enlace y una *especificación de presentación*. El especificador denota un componente y un ancla dentro de ese componente que puede servir como punto final de un enlace. La dirección indica si el punto final especificado actúa como origen o destino del enlace. Sus valores posibles pueden ser DE, A, BIDIRECCIONAL y NINGUNO (para representar botones). De esta forma, si un componente enlace contiene más de dos especificadores con dirección A, estamos representando un enlace n-ario. La especificación de presentación permite definir como va a ser presentado el enlace al usuario. Hablaremos de estas especificaciones más adelante. La Figura 2.4 ilustra un hipertexto desde el punto de vista del nivel de composición interna, supuesto que se elige una representación basada en lenguajes de marcado. También podía haberse optado por una representación gráfica, pero se ha optado por una textual.

```
<hipertexto>
        <componente>
                <tipo> texto </tipo>
                <UID> t23 </UID>
                <datos> Este texto ...  </datos>
                <ancla>
                   <id> 1 </id>
                   <localización> 234, 7 </localización>
                </ancla>
        </componente>
        <componente>
                <tipo> texto </tipo>
                <UID> t12 <UID>
                <datos> Texto ...  </datos>
                <ancla>
                   <id> 1 </id>
                   <localización> 1, 1 </localización>
                </ancla>
        </componente>
        <componente>
                <tipo> enlace <tipo>
                <UID> e88 </UID>
                <especificador>
                        <componente_UID> t23 </componente_UID>
                        <ancla_id> 1 </ancla_id>
                        <direccion> DE </dirección>
                </especificador>
                <especificador>
                        <componente_UID> t12 </componente_UID>
                        <ancla_id> 1 </ancla_id>
                        <direccion> A </direccion>
                </especificador>
        </componente>
</hipertexto>
```

Figura 2.4 Ejemplo de enlace (`e88`) entre el componente `t23` (origen) y el componente (`t12`). Las anclas identifican las posiciones de estos textos donde se origina y recibe el enlace, respectivamente.

Los componentes descritos hasta este momento son lo que el modelo denomina un *componente base*, pero en el modelo los componentes son entidades formadas por un componente base junto a alguna información asociada, denominada *información del componente*. Esta información del componente describe información del componente distinta de su contenido. En particular la información del componente contiene una secuencia de anclas que indexan al mismo, una especificación de presentación que indica como se va a presentar el componente al usuario, y un conjunto de pares atributo/valor. Estos pares representan propiedades del componente.

Además del modelo de datos, el nivel de almacenamiento define un conjunto de operaciones utilizadas para acceder y/o modificar el hipertexto. Estas operaciones permiten añadir un componente al hipertexto, eliminarlo, modificar los contenidos del componente, o de su información. También hay operaciones para obtener un componente dado su UID o cualquier especificador que pueda ser resuelto al componente (es decir, las funciones de acceso y resolución). Finalmente hay dos operaciones que pueden ayudar a determinar los enlaces entre los componentes. Una permite obtener a partir de un hipertexto y del UID de un componente, el conjunto de UIDs de los componentes enlace cuyos especificadores se resuelvan al UID del componente. La otra devuelve a partir de un hipertexto, el UID de un componente, y el identificador del ancla, el conjunto de UIDs de los componentes enlace que se refieren al ancla. Todas las operaciones mantienen los invariantes de un hipertexto. En particular esto exige:

- La función de acceso debe ser una función invertible de UIDs a componentes. Esto implica que cada componente debe tener un UID.
- La función de resolución debe ser capaz de producir todos los UIDs válidos.
- No hay ciclos en la relación componente/subcomponente.
- El identificador del ancla de un componente debe coincidir con el valor de los especificadores que se resuelven en el componente.
- El hipertexto debe ser *enlace-consistente*, es decir, los especificadores de componentes de cada especificador de un enlace deben resolverse a componentes existentes. De esta forma cuando se elimina un componente se deben eliminar todos los enlaces que lo referencien.

El nivel de ejecución. Esquema navegacional

El concepto fundamental en el nivel de ejecución es el de *instanciación* de un componente (crear un ejemplar). Una instanciación es una presentación del componente al usuario, pudiendo haber más de una instanciación simultánea para cualquier componente. Con tal fin, a cada instanciación se le asigna un único identificador de instanciación, IID (*Instantiation IDentifier*) dentro de cada sesión. La instanciación de un componente también conlleva la instanciación de sus anclas. Un ancla instanciada se denomina *marcador de enlace*, indicando que ese ancla es origen de un enlace. Para que la noción de marcador de enlace tenga cabida dentro del modelo, una instanciación realmente es una entidad compleja formada por una *instanciación base* junto con una secuencia de marcadores de enlaces y funciones, traducción de estos marcadores a las anclas que instancian. Por tanto una instanciación base es una primitiva en el modelo que representa algún tipo de presentación del componente al usuario.

En cualquier momento, el usuario de un hipertexto puede ver o editar cualquier número de instanciaciones de componentes. El nivel de ejecución incluye una entidad denominada *sesión* que sirve para gestionar la traducción entre componentes y sus instanciaciones. De esta forma, cuando un usuario quiere acceder a un hipertexto, abre una sesión sobre ese hipertexto. El usuario crea instanciaciones de los componentes del hipertexto a través de una acción denominada *presentación* del componente. El usuario puede editar la instanciación, modificar el componente (*realizar* los cambios), y destruir la instanciación (*ocultar* - del inglés *unpresenting* - el componente). Si el usuario borra un componente en alguna de las instanciaciones, entonces se eliminan automáticamente el resto de instanciaciones. Cuando el usuario termina de interactuar con el hipertexto se cierra la sesión. Es en este punto donde se utilizan las especificaciones de presentación con el fin de caracterizar la interfaz gráfica de usuario que se proporciona a los contenidos, así como esta varia en función de la selección de enlaces. A partir de ahora, y a lo largo de esta tesis, denominaremos a la interfaz gráfica de usuario que se impone sobre los contenidos, y las relaciones navegacionales establecidas entre los elementos de esta interfaz *esquema navegacional* de la aplicación hipermedia.

El núcleo del modelo de ejecución es la función *instanciadora* (del inglés *instantiator*). La entrada de esta función es el UID de un componente y una especificación de presentación (evidentemente en este nivel también hay versiones de la función de acceso y resolución del nivel anterior). La función devuelve una instanciación del componente como parte de la sesión. La especificación de presentación es primitiva en el modelo, y su finalidad es indicar como presentar el componente durante la instanciación. Nótese que el propio componente ya contiene una especificación de presentación. Es tarea de la función instanciadora el utilizar una especificación u otra. La función instanciadora es la base fundamental de la operación *presentar componente*. Esta función toma un especificador de componente (junto con una sesión y una especificación de presentación) e invoca a la función instanciadora utilizando el UID derivado de resolver el especificador. La función presentar componente es el núcleo de la función *seguir enlace*. Esta función toma el IID de una instanciación junto con un marcador de enlace contenido en esa instanciación. Presenta los componentes que

son los puntos final destino (es decir, los puntos final cuyos especificadores tienen dirección A o BIDIRECCIONAL) de todos los enlaces que tienen como punto final el ancla representada por el marcador de enlace. En el caso en que todos los enlaces son binarios, esto es equivalente a seguir el enlace del marcador de enlace a su destino. El resultado de seguir el enlace es una presentación de su componente destino y ancla.

2.2.1.2 Ejemplo

Veamos caracteriza el modelo Dexter al ejemplo del apartado 2.1.1. En particular, este modelo solo es capaz de representar el nivel de almacenamiento de este tipo de aplicaciones, siendo imposible caracterizar la interfaz de usuario aquí propuesta (a no ser claro, que se proporcione un formalismo concreto para las especificaciones de presentación, cosa que el modelo Dexter no hace). Además, la existencia de contenidos dinámicos imposibilita la representación total de los contenidos. Por otro lado, las especificaciones de presentación añadidas a los componentes pueden ser lo suficientemente potentes como para representar esta interfaz de usuario, pero el propio modelo no proporciona ni restringe la naturaleza de dichas especificaciones, por lo que no podemos dar una especificación de presentación basada en el modelo. Nótese que esto no es una deficiencia del modelo, sino el resultado de intentar utilizar un modelo de referencia como un modelo hipermedia puro.

Si obviamos la interfaz de usuario, el nivel de composición interna restringido a los componentes estáticos está formado por el conjunto:

{ biografía, españa, madrid, lepe, estudios, docencia, isg1, isg2, isg91 }

La aplicación estaría representada por el siguiente hipertexto, correspondiente al nivel de almacenamiento. Nótese como las especificaciones de presentación son meros comentarios, muy lejanos a la potencia descriptiva proporcionada por sistemas de representación hipermedia con semántica de presentación formalmente descrita como pudiera ser la red de Petri del modelo Trellis. El modelo propuesto en esta tesis incluye una semántica de presentación que permite generar automáticamente la presentación del los contenidos. Por lo demás, en este ejemplo optamos por la representación utilizada por los autores en su artículo [Halasz 94].

Aunque la representación total de la aplicación puede encontrarse en el Apéndice D (hemos decidido colocarla en un apéndice por su extensión, cercana a seis páginas) veremos aquí algunos elementos. En particular, hay dos tipos fundamentales de componentes. Los contenidos y los enlaces. Un ejemplo de contenido puede ser el que describe a la biografía, y la foto de españa..

```
<componente>
        <tipo> texto </tipo>
        <UID> idBiografía </UID>
        <datos> ... </datos>
        <ancla>
           <id> 0 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 1 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 2 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 3 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 4 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 5 </id>
           <localización> ... </localización>
        </ancla>
```

```
        <ancla>
           <id> 6 </id>
           <localización> ... </localización>
        </ancla>
        <presentación> ventana 1 / panel 1 </presentación>
</componente>
<componente>
        <tipo> foto </tipo>
        <UID> idEspaña </UID>
        <datos> ... </datos>
        <ancla>
           <id> 0 </id>
        </ancla>
        <ancla>
           <id> 1 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 2 </id>
           <localización> ... </localización>
        </ancla>
        <presentación> ventana 2 / panel 1 </presentación>
</componente>
```

De la misma forma, un componente que especifica un enlace entre la biografía y la foto es:

```
<componente>
        <tipo> enlace <tipo>
        <UID> e1 </UID>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idEspaña </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
```

Es decir, primero especificamos que vamos a tener tantos componentes como contenidos, y luego relacionamos esos contenidos mediante enlaces. Cada componente incluye la información del componente, es decir, sus anclas y su especificación de presentación. Comentemos esto en detalle. La forma de especificar enlaces nos ha obligado a utilizar un componente por cada contenido, pero si el concepto de componente estaba ligado al de nodo ¿cómo se las va a ingeniar el sistema para presentar en una misma pantalla tres nodos distintos? La respuesta es bien sencilla. Depende de la implementación. En principio el modelo Dexter no indica que se puedan tener presentaciones simultáneas tras seguir un enlace, es decir, presentaciones del estilo *frame* (*contexto* Amsterdam sería más correcto) como las propuestas en esta tesis. Pero si el sistema hipermedia tiene un nivel de ejecución lo suficientemente avanzado como para entender estas especificaciones de presentación no habría problema.

2.2.1.3 Discusión y comparativa

El modelo Dexter es un buen intento de unificar la terminología y los conceptos de los sistemas hipermedia de antes de los años 90. Vamos a comparar este modelo según las características enumeradas.

Separación de contenidos, navegación y presentación. De indudable valor es la separación de contenidos y de la estructura navegacional de la aplicación, es decir, de la estructura de nodos (componentes) y enlaces entre estos. Esta separación ya aparecía en los modelos de hipergrafos, y de Trellis (anteriores cronológicamente), pero el concepto de ancla para relacionarlo con el nivel de almacenamiento no estaba tan desarrollado en estos modelos.

Contexto. Como ya hemos comentado, el modelo Dexter no considera explícitamente la noción de contexto, pero una extensión de las especificaciones de presentación y del nivel de ejecución permitiría incluir este concepto.

Sincronización. En este aspecto podemos hacer la misma consideración que en el anterior.

Formalización de la semántica de navegación. Una desventaja del modelo es el no proporcionar una formalización para representar la componente navegacional del modelo (a pesar de contar con una formalización de sus estructuras de datos, y del dominio y recorrido de sus funciones en el lenguaje de especificación Z). Como hemos visto la semántica navegacional viene determinada totalmente por el nivel de almacenamiento y las especificaciones de presentación, las cuales no son cubiertas por el modelo. El modelo propuesto en esta tesis puede entenderse como una formalización del modelo Dexter junto a una ampliación del mismo. Podemos resumirlo en la siguiente frase: "El modelo Pipe: añadiendo tiempo, contexto, contenidos dinámicos y semántica de navegación al modelo Dexter", frase inspirada en el título del artículo de Lynda Hardman [Hardman 94].

Organización relacional de los contenidos. Este modelo no supone ninguna organización interna de los contenidos (aunque de hecho esta exista), ni entre los contenidos más allá de la estructura hipertextual. Esto lejos de ser una desventaja supone un gran acierto, ya que como el modelo indica en el campo de las aplicaciones hipermedia el intentar restringir el formato concreto de los contenidos puede ser un error. Es el nivel de almacenamiento el único que induce una estructura de grafo a partir de los enlaces (relaciones) existentes entre los contenidos.

Gestión explícita de eventos. Esta es una característica muy avanzada, que únicamente incluyen los sistemas de referencia hipermedia más modernos y que no presenta el modelo Dexter.

Generación dinámica de contenidos. Como ya hemos comentado, el modelo Dexter no es capaz directamente de representar la generación dinámica de contenidos, ya que cuenta con función de acceso, pero no de generación. Ampliando dicha función si que sería posible caracterizar los contenidos dinámicamente generados. Nótese que también sería necesario ampliar las características del modelo para poder representar anclas entre estos contenidos dinámicamente generados.

Gestión de usuarios y seguridad. Aunque la gestión de seguridad no está presente en este modelo, si que tiene en cuenta (de cierta forma) a los usuarios a través del concepto de sesión. Ahora bien no especifica de que forma se puede restringir el acceso y/o modificación en función del usuario.

Distinción de enlaces a nivel contenidos y esquema navegacional. Este modelo como la mayoría no distingue entre enlaces a nivel de contenidos ni enlaces a nivel de navegación (aunque evidentemente los enlaces de contenidos se presenten a un nivel navegacional mediante las especificaciones de presentación). Una opción factible sería el permitir que los contenidos tuvieran su propio esquema de enlaces que después se tradujera de distintas formas al nivel de almacenamiento. De los sistemas de referencia analizados en este capítulo, solo el modelo OOHDM presenta esta característica.

Técnicas orientadas a objetos. El modelo original data de 1990. En aquella época las técnicas orientadas a objetos no estaban los suficientemente extendidas como para considerar su inclusión en sistemas hipermedia puros. Nótese que una posible extensión del modelo podría ser el considerar como contenidos admisibles diagramas orientados a objetos que modelasen pequeñas aplicaciones dentro de la aplicación hipermedia. Si considerásemos el código que implementa a dichos diagramas como especificaciones de presentación, y suponiendo una extensión del nivel de ejecución, tendríamos la posibilidad de cubrir con el modelo Dexter una vertiente no considerada por los autores.

Como podemos ver el modelo Dexter es un modelo limitado en muchas características, pero su alto nivel de generalidad puede hacerle englobar a otros muchos sistemas de referencia. Además es el único sistema de referencia que se preocupa de modelar de forma abstracta sistemas hipermedia, proporcionando para tal fin el nivel de ejecución. Por lo demás es un modelo que suelen referenciar la mayoría de autores de sistemas de referencia hipermedia, y que sirvió como base para extensiones como la propuesta en el modelo Amsterdam.

2.3 Sistemas de representación con sincronización y contexto

Dentro de este apartado nos encontramos con distintos sistemas de representación hipermedia, aunque en la práctica solo veremos al modelo Amsterdam. Hemos preferido incluir solo este modelo, y clasificar el resto de sistemas de representación según otras características más distintivas.

2.3.1 El modelo Amsterdam

2.3.1.1 Presentación

Introducción

El modelo Amsterdam [Hardman 93], [Hardman 94] surge como una evolución de dos modelos ya existentes, el modelo de hipertexto Dexter, y el modelo multimedia CMIF [Hardman 93]. Respecto al primero intenta paliar las deficiencias que vimos en el apartado anterior (sobre todo sincronización y contexto). Respecto al segundo intenta ampliar la reproducción de medios en un entorno hipertextual. Además el modelo incluye definiciones tan interesantes como la de canales, que son dispositivos abstractos de reproducción. A continuación estudiaremos el modelo Amsterdam en base a sus características más destacadas.

Composición de distintos medios

El modelo Amsterdam, al igual que el modelo Dexter, considera la posibilidad de componer distintos medios. Ahora esta composición se hace desde dos puntos de vista el *temporal* y el *espacial*.

La *composición espacial* toma como punto de partida los componentes del modelo Dexter pero con importantes variaciones. Ahora los elementos compuestos no pueden tener contenidos, estando relegada esta tarea a los componentes atómicos. De esta forma los componentes compuestos son meros portadores de estructura, que se impone a los contenidos (los cuales se encuentran en los componentes atómicos). Esta aproximación proporciona una visión más clara del concepto de composición, y evita (como veremos) el problema de donde localizar los contenidos en la jerarquía de presentación (siempre están en las hojas).

En el modelo Amsterdam, un componente compuesto especifica a los hijos que forman el componente. Estos pueden ser componentes atómicos (de medios estáticos o dinámicos) o a su vez, otros componentes compuestos. Los componentes compuestos pueden ser de dos tipos paralelos o de elección. Un *componente compuesto paralelo* indica que cuando se acceda a dicho componente, todos sus hijos deben ser presentados al usuario. Un *componente compuesto de elección* indica que al menos uno de sus hijos debe ser presentado al usuario. La idea de los componentes de elección es que los hijos del nodo están altamente relacionados y que estarán enlazados uno con otros, presentándose el contenido de un hermano cuando se acceda a él a través de un enlace. Los hijos de los componentes compuestos se identifican vía identificadores únicos. Los componentes compuestos son la base, como veremos más adelante, de la idea de contexto. La Figura 2.5 ilustra la noción de componente compuesto.

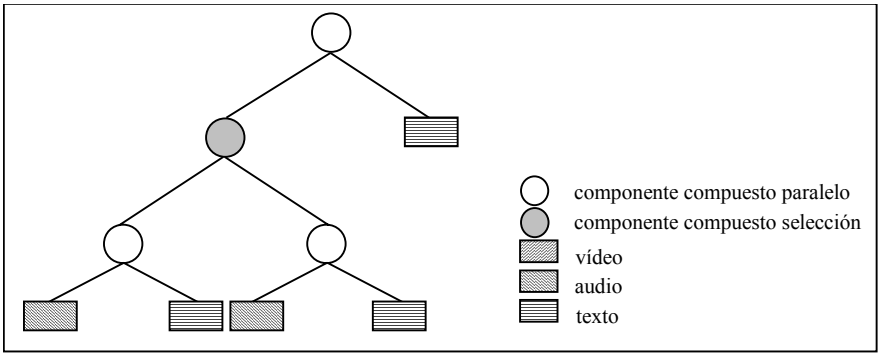

Figura 2.5 Un componente compuesto del modelo Amsterdam.

Con respecto a las *relaciones temporales,* las relaciones de sincronización para cada hijo se establecen en base a sus padres o hermanos, y se deducen de la estructura de composición paralela, o pueden ser proporcionadas explícitamente por el autor. La duración del componente padre depende de las duraciones de sus hijos y de las relaciones temporales. Las relaciones temporales respecto al padre pueden indicar que un nodo hijo se muestra al usuario: cuando empieza el padre, tiempo después de que empiece el padre, tiempo después de que acabe el padre, o termina cuando termine el padre. Con respecto a los hermanos las posibilidades de sincronización son idénticas.

Canales y combinación de componentes

Los *canales* son dispositivos abstractos para la reproducción de elementos multimedia. Estos pueden ser, por ejemplo, una ventana en una pantalla, o una salida de audio (en este apartado utilizamos la notación original del modelo). Precisamente los *nodos contenedores* del modelo propuesto en esta tesis recogen este concepto. Los canales incluyen información de presentación por defecto, por ejemplo, la fuente y el estilo para un canal de texto, o volumen para un canal de audio. Es obligatorio asociar los componentes atómicos con canales, ya que la única forma de reproducir cualquier información es a través de su canal correspondiente. El número de canales en un documento no se restringe, y cuando este se reproduce los canales se asocian a dispositivos físicos de salida. Proporcionan una visión de la aplicación independiente de la estructura de la misma.

El modelo también permite asociar canales a tareas. Por ejemplo todos los encabezados de las pantallas pueden asociarse a un "canal superior" con una fuente y estilo predeterminados. De la misma forma, la información principal puede mostrarse en un canal central, y los enlaces a otras partes del documento se muestran en canales en la parte inferior de la pantalla. Además los canales son usados para definir los estilos de presentación de los componentes atómicos que se van a reproducir por un determinado canal. Estos se definen de forma global (a una colección de documentos), y son referenciados por cada componente atómico. Evidentemente, para cada tipo de datos debemos disponer de un canal adecuado, pudiendo utilizar distintos canales para un mismo componente (un mismo texto puede aparecer en distintas ventanas según la pantalla, por ejemplo). La única restricción es que un mismo componente atómico no puede ocupar más de un canal simultáneamente.

El modelo Amsterdam permite además combinar distintos elementos compuestos, con una única restricción: el componente compuesto resultante deberá poder ser reproducido utilizando los recursos existentes. Esta restricción puede ser no ser satisfecha si se integran canales incompatibles (por ejemplo dos canales de audio simultáneos), o se intenta utilizar simultáneamente el mismo canal (por ejemplo, dos ventanas que se superpongan). Por tanto cuando se combinen componentes deben resolverse los conflictos entre recursos. Esta información se puede deducir de los canales e información temporal utilizados por los componentes atómicos descendientes, y las relaciones de sincronización existentes entre ellos.

Relaciones temporales avanzadas

El principal mecanismo para representar relaciones temporales en el modelo Amsterdam es elemento compuesto. La sincronización entre componentes atómicos puede ser de dos tipos de *grano grueso* y de *grano fino*. La primera es la que se establece en base a relaciones temporales entre un nodo y su padre o hermanos (la vimos en el apartado de composición). La de grano fino se establece entre descendientes no hermanos, y se representa a través de los *arcos de sincronización*. Estos arcos representan una relación temporal similar a las de grano grueso, pero entre descendientes de distintos niveles.

Por ejemplo tiempo después de que aparezca un texto debe empezar a sonar una música que se encuentra en otro nivel del componente compuesto. Para expresar estas relaciones utilizaremos un arco de sincronización entre ambos componentes atómicos. Estos arcos solo pueden establecerse entre elementos de un mismo componente compuesto. Nótese que estos arcos no se utilizan como un tipo de enlace. Es una restricción que el sistema de ejecución debe soportar para reproducir los componentes. La Figura 2.6 ilustra un arco de sincronización entre componentes atómicos de un componente compuesto.

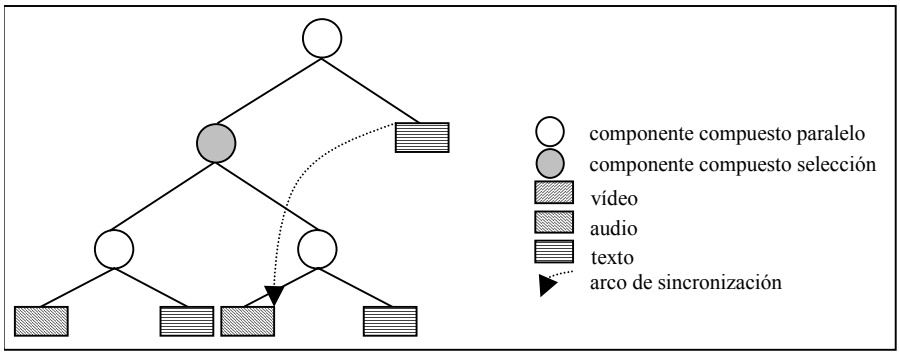

Figura 2.6 Arco de sincronización

Contexto para los enlaces

El modelo Amsterdam, como no podía ser de otra manera, también soporta enlaces navegacionales del estilo Dexter. La Figura 2.7 ilustra un enlace de este estilo entre componentes.

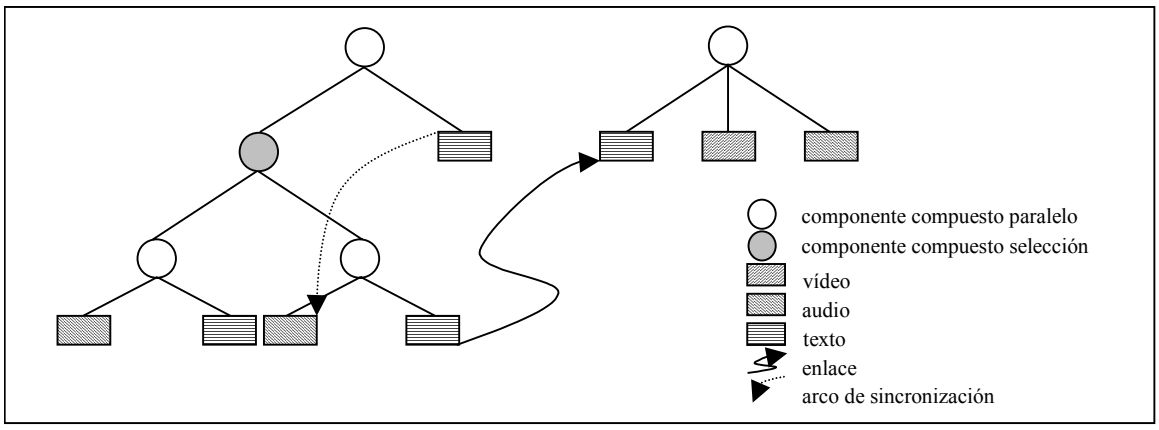

Figura 2.7 Un enlace entre componentes

El modelo Dexter no indicaba explícitamente que información se sigue mostrando y cual se actualiza en una pantalla una vez que se selecciona un enlace. La idea del modelo Dexter es presentar un componente entero cada vez que se seleccione un enlace. Esta presentación se realiza acorde al sistema que implemente al modelo, las especificaciones de presentación, y las implementaciones de las funciones para seguir enlaces y presentar. El modelo Amsterdam define el *contexto* origen o destino de un enlace como la parte de la estructura del documento que se ve afectada cuando se selecciona el ancla como origen, o se presenta como destino del enlace. La ventaja de los contextos es que solo una parte de la estructura de un componente compuesto se ve afectada cuando se selecciona un enlace. Los nodos de la presentación más arriba en la jerarquía se mantienen en pantalla mientras que solamente los nodos de niveles inferiores son reemplazados. Nótese que al combinar el concepto de contexto con el de canal se obtiene un mecanismo más potente que el de frame HTML. La Figura 2.8 ilustra este concepto. En esta figura cuando se selecciona el enlace con origen en el contenido textual solo se reemplazan los componentes de la parte B, mientras que la parte A se mantiene en pantalla. Es decir, se mantiene el contexto. La figura muestra solo dos niveles, pero el modelo no impone ninguna restricción sobre la profundidad.

Por tanto con el fin de soportar la noción de contexto debemos especificar para cada ancla si es origen o destino de un enlace, la posición espacial de la misma, y el contexto, es decir, la parte de la presentación que será afectada cuando se siga el enlace o cuando se llegue a él. El contexto se especifica en el modelo Amsterdam asignando un identificador de componente (probablemente compuesto) a cada ancla. Estas anclas pueden referenciar a otras anclas localizadas en descendientes del mismo componente compuesto. Las anclas estarán localizadas en los componentes atómicos, que contienen las descripciones de las mismas en función del tipo de datos al que estén sujetas.

Nótese que por tanto en las anclas se especifica tanto el *contexto origen* como el *contexto destino*. Es decir, una vez que se ha establecido el contexto asociado con los finales de un enlace, podemos utilizar distintas opciones de presentación para seguir un enlace (aumentando las posibilidades de reproducción respecto al

comportamiento por defecto descrito anteriormente). El contexto origen puede ser mantenido o reemplazado. Si se reemplaza, entonces el contexto destino será mostrado en el lugar que se encontraba el contexto origen (esto requiere mecanismos de control para garantizar que el nuevo contexto es compatible con el anterior). Si el contexto destino se mantiene, entonces la presentación puede continuar, o puede pararse.

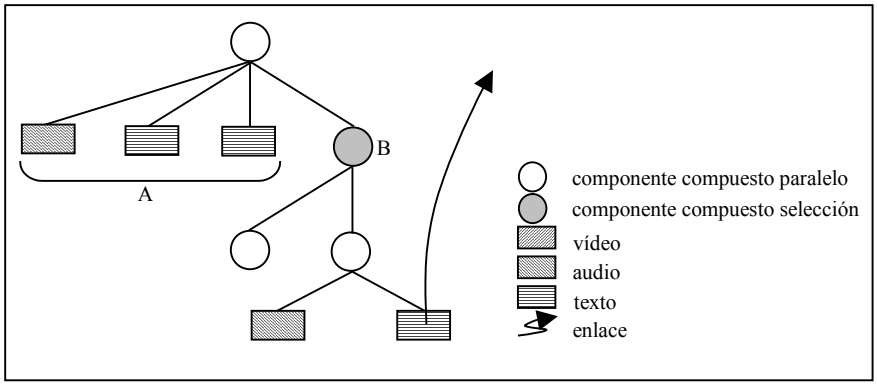

Figura 2.8 Contexto para los enlaces

2.3.1.2 Ejemplo

Veamos como representa el modelo Amsterdam el ejemplo que venimos manejando. En este caso el modelo Amsterdam no contempla la posibilidad de representar los contenidos dinámicamente generados, así que nos restringiremos a los componentes estáticos. En este caso necesitamos al menos tres canales de reproducción correspondientes a los paneles. Llamemos a estos canales p1, p2, y p3. Asignamos la biografía al canal p1, la foto de españa, los estudios y la docencia al canal p2, y finalmente madrid, lepe, isg1, isg2, e isg91 al canal p3.

En este caso, necesitamos un componente compuesto para representar la primera pantalla, y otro para representar la segunda pantalla. La Figura 2.9 recoge estos componentes.

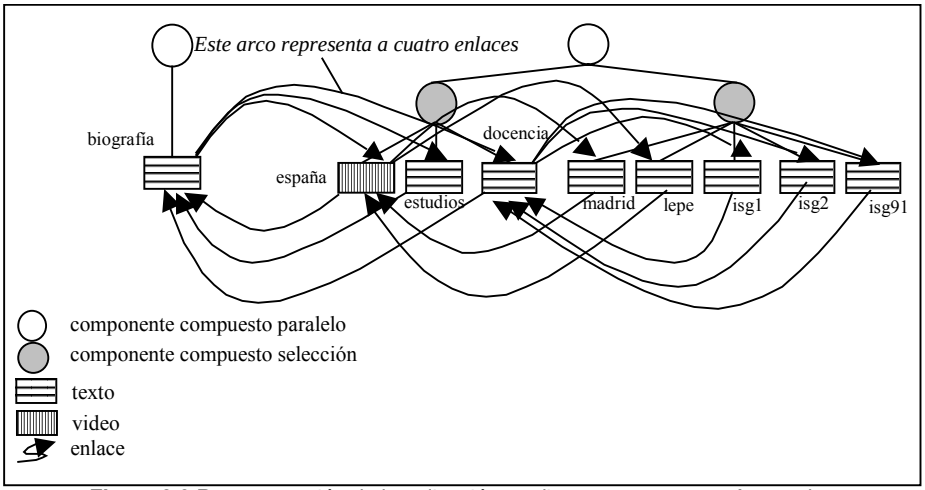

Figura 2.9 Representación de la aplicación mediante componentes Amsterdam.

En el primer componente se muestra la biografía. En el segundo componente se va a mostrar de forma paralela la selección que provenga de la biografía, y la selección que provenga del contenido seleccionado por la biografía.

Como podemos ver el modelo Amsterdam es capaz de representar sin problemas la noción de *frame*. Solamente se empieza a detectar un inconveniente: la presencia explícita de todos los enlaces. En este caso no hay demasiado problema ya que solo hay dos pantallas, tres ventanas y diecinueve enlaces, pero parece que se hace necesaria una ocultación de algún tipo de los enlaces de contenidos, indicando únicamente los enlaces

existentes entre canales. Veremos en el Capítulo 5 como el modelo propuesto en esta tesis resuelve el problema al separar los enlaces de contenidos de su interpretación en el esquema navegacional.

2.3.1.3 Discusión y comparativa

El modelo Amsterdam supone una verdadera extensión al modelo Dexter. Provee las nociones de sincronización y contexto, imprescindibles en las aplicaciones hipermedia actuales. Veámoslo en la comparativa.

Separación de contenidos, navegación y presentación. Evidentemente la separación de contenidos de la estructura navegacional se va a presentar en todos los sistemas de representación hipermedia (salvo en el modelo HAM), ya que es una consideración trivial: los contenidos existen, con independencia de su inclusión en una red de hipertexto. Ahora bien, el nivel de almacenamiento se ve sustancialmente modificado respecto al modelo Dexter. El nivel de almacenamiento Dexter era equivalente a un grafo dirigido, donde los componentes (compuestos o atómicos) aparecían en los nodos. Ahora esta estructura de grafo "se rompe". En efecto, la noción de contexto actúa como una desvirtualización del grafo, ya que cuando seleccionamos un enlace de un contexto de un componente compuesto a otro contexto de otro componente compuesto, lo que se visualiza no es un componente asociado a un nodo, sino parte de dos componentes en dos nodos distintos. Además la presentación no se separa del nivel de almacenamiento, es más, depende casi exclusivamente del mismo. En este sentido, el modelo Amsterdam posee una herramienta (Grins), capaz de generar aplicaciones hipermedia en base a los componentes del modelo. Esta herramienta genera una representación alternativa del modelo (o de la aplicación, como prefiera entenderse) en un lenguaje XML concreto: SMIL. El nivel de presentación viene representado por las especificaciones de presentación asociadas a los canales, y por la estructura jerárquica inducida por el componente compuesto.

Contexto. La aportación de contexto en el campo de la hipermedia es uno de los mayores avances desde nuestro punto de vista. El contexto rompe totalmente la estructura de grafo, y proporciona una mayor potencia a la hora de describir la interfaz de usuario. Como hemos visto en el ejemplo, el modelo Amsterdam utiliza una notación visual para describir contextos. La ventaja proporcionada por esta descripción gráfica puede convertirse en un serio inconveniente si existe un gran número de enlaces y restricciones de sincronización entre los contenidos de la aplicación hipermedia. En estos casos, la legibilidad de la representación disminuye notablemente.

Sincronización. Este es uno de los aspectos más potentes del modelo Amsterdam. Permite sincronización de grano grueso y de grano fino. Con mucho una de sus más brillantes características.

Formalización de la semántica de navegación. Al igual que el modelo Dexter, el modelo Amsterdam no proporciona ninguna formalización explícita de su semántica de navegación (a pesar de contar con una formalización de sus estructuras de datos, y del dominio y recorrido de sus funciones en el lenguaje de especificación Object-Z). Además ahora no es nada trivial representar que sucede cuando se sigue el enlace de un contexto a otro componente. El modelo propuesto en esta tesis proporciona una formalización para representar contextos, y mostrar que sucede cuando se sigue un enlace entre contextos, es decir, la semántica de navegación. En el modelo Amsterdam la semántica de navegación viene determinada por el componente compuesto, y las relaciones jerárquicas y temporales impuestas sobre este. En la práctica puede resultar bastante complejo intentar visualizar esta semántica de navegación sin una herramienta que nos ayude. De hecho el modelo cuenta con dichas herramientas. Nótese que a pesar de la falta de formalización el modelo si que cuenta con una semántica de navegación por defecto, pieza clave de los sistemas de desarrollo basados en el modelo (Capítulo 3).

Organización relacional de los contenidos. Al igual que el modelo Dexter, este modelo no considera ninguna organización de los contenidos. Solamente supone que cada contenido es susceptible de contener anclas, mecanismo del que se obvia una descripción en profundidad por las mismas razones que el modelo Dexter.

Gestión explícita de eventos. En principio el modelo Amsterdam no considera ninguna gestión de eventos. La razón se debe a que el único evento al que es capaz de responder el sistema es a la invocación de enlaces. Podría discutirse si un sistema de referencia hipermedia debe ser responsable de representar que sucede

cuando se produce un evento que no es de este tipo. De no ser capaz se mantiene más simple, pero menos potente. De lo contrario pierde su simplicidad en aras de una mayor potencia representativa.

Generación dinámica de contenidos. El modelo es incapaz de caracterizar contenidos dinámicamente generados, ya que el peso estructural recae sobre la noción de componente compuesto, y la generación dinámica de contenidos provocaría la generación dinámica de componentes compuestos.

Gestión de usuarios y seguridad. El modelo Amsterdam no considera nada acerca de gestión de usuarios ni de seguridad.

Distinción de enlaces a nivel contenidos y esquema navegacional. El modelo no considera una distinción entre ambos niveles de enlaces. Esta es la causa por la cual, cuando se tiene un componente compuesto con ciento diez enlaces es necesario mostrarlos todos. Además, según nuestro criterio, la visualización del esquema navegacional como porciones de diversos componentes compuestos puede resultar bastante compleja en la práctica (e inabordable sin soporte CASE). El modelo propuesto en esta tesis optará (al igual que OOHDM) por una separación de ambos niveles de enlaces, simplificando así la representación gráfica del esquema navegacional de la aplicación, y facilitando su legibilidad.

Técnicas orientadas a objetos. El modelo no incluye técnicas orientadas a objetos, ya que supone que el contenido de las aplicaciones hipermedia será: texto, audio, gráficos o vídeo (básicamente). Es decir, dentro de una aplicación hipermedia no se considera que pueda haber otras aplicaciones ejecutándose, así que no necesita representar nada más de lo que representa. Por esta razón no incluye ningún formalismo para representar este tipo de aplicaciones internas.

2.4 Sistemas de representación con semántica de navegación formalmente definida

En esta sección analizaremos los sistemas de representación que proporcionan un formalismo explícito para representar la estructura navegacional de la aplicación y su semántica navegacional. La máquina abstracta de hipertexto no considera la noción de contexto, y utiliza un grafo para representar la aplicación. El modelo de hipergrafos, si considera parcialmente la noción de contexto y utiliza un hipergrafo para conseguirlo. Finalmente, el modelo Trellis utiliza una aproximación similar, pero en vez de un hipergrafo utiliza una red de Petri. La formalización de la semántica de navegación nos permitirá describir sin ningún tipo de ambigüedad que sucede en el hipertexto cuando se activa un enlace. Además esta es la base para la construcción de entornos que sirvan para generar aplicaciones hipermedia (o prototipos avanzados de estas) en base a las descripciones proporcionadas por los sistemas de representación hipermedia [Heath 00]. Nótese que otros modelos como Amsterdam también presentan esta característica a pesar de carecer de una formalización de su semántica de navegación. Podemos entender que en el caso de Amsterdam, la formalización de la semántica navegacional es el código de las funciones que actúan sobre la descripción XML (es decir, sobre una instancia concreta de la DTD de SMIL) para generar la aplicación. Desde nuestro punto de vista es preferible una caracterización más declarativa, que proporcione una descripción independiente de lenguajes de programación y tecnologías concretas de implementación. El único requisito (que ni mucho menos es un inconveniente, desde nuestro punto de vista) es la formalización del propio sistema de representación hipermedia. Este será el camino elegido por el modelo hipermedia propuesto en esta tesis doctoral.

2.4.1 La máquina abstracta de hipertexto (HAM)

2.4.1.1 Presentación

Introducción

La máquina abstracta de hipertexto, o HAM (*Hipertext Abstract Machine*) [Campbell 88], es el sistema de representación más antiguo aquí presentado (1988). Aunque no es un modelo hipermedia, ya que únicamente intenta modelar el nivel de almacenamiento de un sistema de hipertexto (es decir, la capa de almacenamiento del modelo Dexter), puede ser considerada una forma de representar sencillas aplicaciones hipertextuales. De hecho las páginas HTML originales se ajustaban perfectamente a esta descripción. El modelo de

almacenamiento HAM se basa en cinco objetos: grafos, contextos (sin relación con el contexto del modelo Amsterdam), nodos, enlaces y atributos.

Los objetos HAM

Un *grafo* HAM contiene contextos, nodos, enlaces, y atributos. Estos objetos se organizan de forma jerárquica. Un grafo es el objeto HAM de mayor nivel. Normalmente contiene toda la información concerniente a una aplicación hipermedia. Un grafo puede contener uno o más contextos.
Un contexto (HAM) es una partición de los datos dentro del grafo. Los contextos puede utilizarse para soportar configuraciones, espacios de trabajo privados, y árboles de historial de versiones. Cada contexto tiene un contexto padre y cero o más contextos hijos. Cuando se crea un grafo, se crea un contexto, raíz del árbol. Un contexto que contiene cero o más nodos y enlaces no depende de la información contenida en su contexto padre.
Un *nodo* contiene datos arbitrarios que pueden ser almacenados como texto o como bloques binarios de longitud fija. Un nodo puede ser clasificado, archivado, recuperado, o ser de agregación (del inglés *append-only*). Cuando un nodo archivado se modifica, se crea una nueva versión del nodo con los nuevos contenidos. Las versiones anteriores de un nodo archivado también pueden ser recuperadas. Cuando un nodo sin archivar es recuperado, los contenidos previos se reemplazan por los nuevos. Cuando se modifica un nodo de agregación, los nuevos contenidos se añaden a los ya existentes. Los contenidos de un nodo pueden ser interrogados con patrones de expresiones regulares introducidas por el usuario. Los nodos se relacionan a través de enlaces.
Un *enlace* define una relación entre un nodo origen y un nodo destino, y puede ser seguido en ambas direcciones. Un enlace *entre contextos* relaciona dos nodos en diferentes contextos, y es útil para compartir datos entre ambos contextos. La generalidad proporcionada por los enlaces de atributos permiten a los escritores de aplicaciones definir su propia noción de tipos de enlaces.
Los contextos, los nodos, o los enlaces, pueden tener asociados *atributos*. Los valores de estos atributos pueden ser cadenas, enteros, reales, o datos definidos por el usuario. Los pares atributo/valor dan semántica a los objetos HAM. Estos pueden representar propiedades de los objetos específicas para la aplicación, o contener información descriptiva sobre el objeto. Los atributos se utilizan también como parte de los filtros.

Historial de versiones, filtros y seguridad

La máquina abstracta de hipertexto proporciona un mecanismo automático de historial de versiones. El historial de versiones de un objeto HAM se actualiza cada vez que se modifica el objeto.

También proporciona un mecanismo de filtrado que permite extraer subconjuntos de objetos HAM procedentes de grafos. Los filtros permiten al usuario especificar predicados de visibilidad, que son expresiones que relacionan los atributos con sus valores. Los filtros solo devuelven los objetos que satisfacen los predicados.

La seguridad de los datos contenidos en el grafo se garantiza a través del mecanismo de lista de control de acceso o ACL (*Access Control List*). El proporcionar una ACL a un objeto es opcional. Una entrada de la ACL está formada por un nombre de grupo de usuarios, y un conjunto de permisos.

Operaciones HAM

Para proporcionar una interfaz simple y consistente, las operaciones HAM se agrupan en siete categorías. Las operaciones dentro de cada categoría se comportan de manera similar, con independencia del objeto sobre el que operen. Las *operaciones de creación* crean nuevos objetos HAM. Las *operaciones de borrado* marcan objetos como borrados, pero mantienen información histórica. Las operaciones de *destrucción* liberan todo el espacio que ocupaba un objeto. Las *operaciones de cambio* modifican los datos asociados con un objeto existente. Las *operaciones de acceso* recuperan datos de objetos existentes. Las *operaciones de filtrado* recuperan selectivamente información de un grafo. Las *operaciones especiales* incluyen funciones como búsqueda de cadenas en los contenidos de un nodo, mezcla de contexto y gestión de transacciones.

2.4.1.2 Ejemplo

La máquina abstracta de hipertexto es demasiado básica como para caracterizar la generación dinámica de contenidos, y un esquema navegacional con contexto como el del ejemplo. Si nos restringimos a los

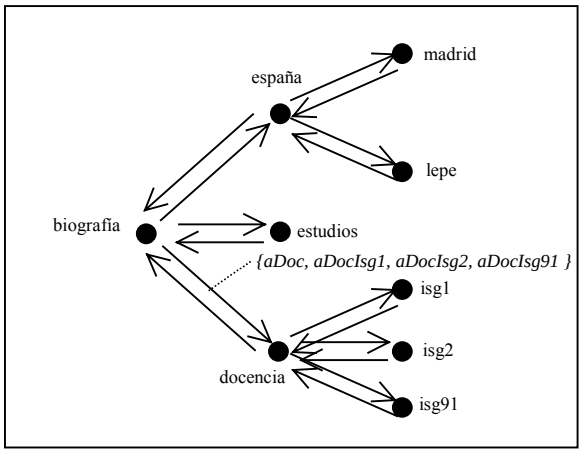

Figura 2.10 Representación HAM de la aplicación

contenidos estáticos, y suponemos que cada contenidos aparece en una pantalla obtenemos un hipertexto como el descrito en la Figura 2.10. En este caso hemos obviado la caracterización matemática del grafo, optando únicamente por su representación gráfica.

2.4.1.3 Discusión y comparativa

Como hemos visto este sistema de representación es extremadamente sencillo, pero no caracteriza el esquema navegacional de las aplicaciones hipermedia, más allá del acceso a un nodo. La mayor ventaja es el formalismo matemático subyacente (un grafo) que sirve para describir la estructura navegacional de la aplicación, y la semántica navegacional de la misma, aunque ambas sean muy básicas.

Separación de contenidos, navegación y presentación. El objetivo de este sistema de representación es describir la capa de almacenamiento del modelo Dexter, así que no tiene ningún tipo de separación entre la estructura navegacional y los contenidos. Aún así se puede suponer implícita. Tampoco especifica ninguna característica presentacional de los contenidos, salvo los atributos de los nodos.

Contexto. Un grafo simplemente, es una estructura demasiado sencilla como para expresar la noción de contexto Amsterdam. La noción de contexto que proporciona el sistema de representación HAM, no deja de ser una mera agrupación de contenidos. El modelo propuesto en esta tesis ampliará esta estructura de grafo con otro grafo ampliado (cuenta con varios tipos de nodos y arcos) utilizado para representar el esquema navegacional de la aplicación. El problema surgirá a la hora de relacionar ambas estructuras. La solución propuesta en esta tesis pasa por la utilización de funciones definidas entre ambas estructuras con el fin de relacionarlas.

Sincronización. El sistema de representación no considera ninguna capacidad de sincronización, ya que únicamente caracteriza el acceso a los contenidos, sin importar cuando sucede.

Formalización de la semántica de navegación. Aunque no formalizado en el sistema de representación, la utilización de un grafo facilita enormemente la visualización del esquema navegacional de la aplicación. Esta idea será tenida en cuenta por el modelo propuesto en esta tesis. La semántica de navegación HAM es muy sencilla, limitándose a mostrar los contenidos asociados a cada nodo, cada vez que se accede a uno de ellos.

Organización relacional de los contenidos. La máquina abstracta de hipertexto no supone ninguna estructura de los contenidos que no sea la proporcionada por el hipertexto.

Gestión explícita de eventos. Al igual que la sincronización, no era necesario modelar este tipo de fenómenos.

Generación dinámica de contendidos. Como ya hemos comentado el sistema de representación es demasiado sencillo como para caracterizar esta posibilidad.

Gestión de usuarios y seguridad. Al ser un sistema de representación hipermedia que caracteriza únicamente el almacenamiento y acceso de los datos, HAM si que presenta una sencilla gestión de usuarios y seguridad, a través de la lista de control de acceso.

Distinción de enlaces a nivel contenidos y esquema navegacional. Como el sistema de representación no concibe contenidos si no son integrados en la red hipertextual, no puede considerar dos niveles de enlaces.

Técnicas orientadas a objetos. El sistema de representación no intenta representar nada ajeno a la posibilidad de acceder a contenidos a través de enlaces. Por eso no incluye ninguna capacidad de representación adicional.

2.4.2 El modelo de hipergrafos

2.4.2.1 Presentación

Introducción

El modelo de hipergrafos [Tompa 89] es una evolución directa del sistema HAM Al igual que este se presenta una formalización (explícita en el caso de hipergrafos) que describe el esquema navegacional, y sirve como base para la semántica de navegación, es decir, que contenidos se muestran al usuario cuando este recorre el hipertexto. En principio, este modelo considera al hipertexto como una base de datos (en formato no relacional) que va a ser representada a través de un hipergrafo. Este hipergrafo aumenta las posibilidades de representar contextos frente a un simple grafo, pero en la práctica se muestra menos potente que el modelo Amsterdam.

Aunque este modelo también puede considerarse un modelo de referencia, al describir el estado de una base de datos hipertextual, la existencia de la estructura de hipergrafo parece más dirigida a caracterizar una aplicación hipermedia, que el sistema que la gestiona. De todas formas, esta consideración no afecta para nada a la clasificación seguida en este capítulo.

Descripción formal de un estado de base de datos

Basado en un hipergrafo dirigido y etiquetado, un estado de base de datos está modelado por la tupla,

$$D = <N, P, R, V, L, E>$$

donde:

- N, es un conjunto finito de *nodos*.
- P, es un conjunto de *páginas* (contenidos de los nodos).
- R, es un conjunto de *lectores* que interpretan las páginas
- V: $N \rightarrow P$, es una *función de valor* que asigna páginas a los nodos
- L, es un conjunto de etiquetas.
- E: $\wp(N) \times \wp(L) \rightarrow \wp(N)$, es un conjunto de *hiperarcos*, siendo $\wp(X)$ el conjunto *partes de* el conjunto X. Basándonos en este conjunto definimos el conjunto Arcos(E),

$$Arcos(E) = \{ (n, n'; l) \mid N' \subseteq N, L' \subseteq L, n \in N', l \in L', n' \in E(N', L') \}$$

  donde (n, n'; l) representa un arco dirigido con origen en el nodo n, destino el nodo n', y etiqueta l. Es decir, cada hiperarco es equivalente a un conjunto de arcos dirigidos etiquetados.

Dentro del modelo de hipergrafos, los usuarios están situados sobre varios nodos simultáneamente, y un usuario tiene acceso concurrente a varios segmentos de la base de nodos hipertextual. Un estado de usuario está formado por el estado de la base de datos, junto con un conjunto $M \subseteq N$, que sirve como un *conjunto de marcadores*, para designar los nodos actuales. La Figura 2.11 ilustra el concepto de hipergrafo. Utilizando el modelo de hipergrafos, la navegación se interpreta de la siguiente forma: para cualquiera de los nodos (incluido todos) marcados en un estado de usuario, un usuario puede:

- Invocar a un lector para que interprete la página asociada con el nodo.
- Examinar el conjunto de etiquetas cuyo conjunto origen incluya al nodo.
- Marcar todos los nodos destino que sean destino de una etiqueta con origen en el nodo marcado. Es decir, si el nodo marcado es n, y la etiqueta es l, entonces marcamos todos los nodos m tal que (n, m; j)∈ Arcos(E).

En el estado de usuario descrito en la Figura 2.11, las páginas 1 y 3 están siendo mostradas simultáneamente (por ejemplo, mostrando un texto y reproduciendo un clip de audio). Si se selecciona la etiqueta detalles se marcarían los nodos asociados las páginas 2, 4 y 5, produciéndose una presentación simultánea de los mismos.

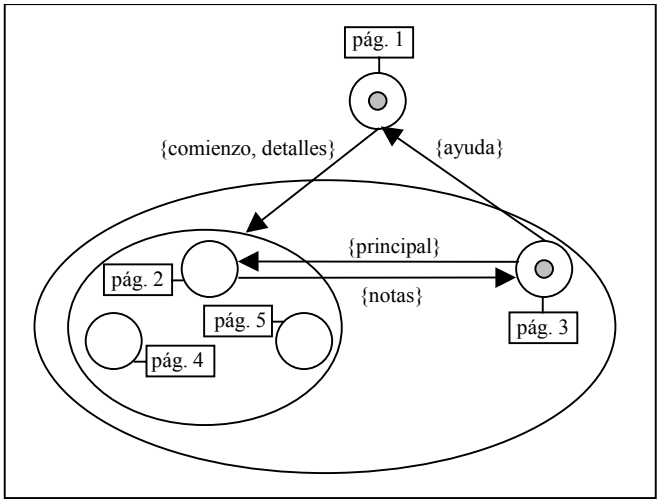

Figura 2.11. Representación de un hipertexto a través de un hipergrafo. El símbolo ○ determina los nodos activos y las elipses el conjunto de nodos de un hiperarco.

El modelo de hipergrafos también considera el concepto de *vista de usuario*. Una vista de usuario permite aumentar o restringir los nodos a los que el usuario tiene acceso. Formalmente una vista se define como una tupla,

$$D' = <N', P', R', V', L', E'>$$

- N', es un conjunto finito de nodos
- P', es un conjunto de páginas (contenidos de los nodos)
- R', es un conjunto de lectores que interpretan las páginas
- V': $(N' \cup N) \rightarrow (P' \cup P)$, es una función de valor que asigna páginas a los nodos, donde

$$V'(n) = \begin{cases} V'(n), \text{ donde esté definida} \\ V(n), \text{ en otro caso} \end{cases}$$

- L, es un conjunto de etiquetas
- E': $P(N' \cup N) \times P(L' \cup L) \rightarrow P(N' \cup N)$, es un conjunto de hiperarcos, donde,

$$(n, m; l) \in Arcos(E') \Leftrightarrow (n, m; l) \in Arcos(E') \vee$$
$$((n, m; l) \in Arcos(E) \wedge \forall n' \in N' \cup N, (n, n'; l) \notin Arcos(E'))$$

es decir, el arco se define en la vista, o fue previamente definido y no se redefine en la vista.

El conjunto de marcadores M en un estado de usuario se define $M \subseteq N \cup N'$. Por lo tanto, una vista permite a un usuario aumentar la base de datos de manera arbitraria (redefinir a las asociaciones de páginas con nodos y las conexiones entre estos). Como el definir el estado de usuario sobre una vista es equivalente a definirlo directamente sobre la base de datos, cualquier operación en la base de datos se puede aplicar a la vista. En particular se puede superponer otra vista, obteniendo vistas en cascada.

Operaciones dentro del modelo de hipergrafos

El modelo de hipergrafos considera al hipertexto como una base de datos (no relacional), y por tanto proporciona una serie de operaciones para gestionar dicha base de datos. Estas operaciones sirven para preguntar sobre un estado, crear vistas y gestionar la base de datos. Representan parte de la semántica de navegación de la aplicación. La lista total de operadores puede encontrarse en [Tompa 89]. Aquí nos centraremos en los de mayor relevancia, y en particular en los de interrogación.

Los operadores de *interrogación* se dividen en varias categorías: los que permiten navegar por la estructura, los que permiten acceder a páginas (leerlas o ejecutarlas como corresponda), los que permiten mostrar la posición en la estructura, y los que controlan los efectos secundarios.
El operador Step (U:EstadoUsuario; const L:etiqueta; const dir: (forward, reverse)), permite navegar por el hipergrafo. Si la dirección es forward, para cada nodo marcado v en el estado de usuario U, se colocan marcadores en todos los nodos n del conjunto N tal que (v, n; L)∈Arcos(E), donde E es el conjunto de hiperarcos en U. De forma similar, se invierte la operación anterior si la dirección es reverse.

El operador MarkAllNodes(U:EstadoUsuario) marca una configuración inicial de usuario en la base de datos.

El operador Execute(const U:EstadoUsuario; const R: lector) aplica el lector sobre todos los contenidos marcados en el estado de usuario.

Los operadores AutoRemove(const avance, finales: booleanos), y AutoShow (const página, caminos: booleanos), afectan a la semántica de navegación controlando los efectos secundarios sobre la misma. Cuando aplicamos el operador Step es posible que los nodos origen permanezcan marcados, o que al contrario, desaparezca su marca. Esto se controla con el operador AutoRemove. El modelo de la expansión del marcador se puede fijar a eliminar (avance = v), o preservar (avance = f). Al igual podemos eliminar los marcadores de los nodos que no son origen para ningún arco etiquetado en E que se vea involucrado en una transición, mediante el parámetro finales. Por otro lado, cuando se accede a un nodo podemos desear que automáticamente se muestre el contenido de la página asociada con ese nodo, y los caminos que parten de él (los cuales se muestran como menús). El operador AutoShow controla estas opciones.

Por ejemplo si en el estado de usuario de la Figura 2.9 aplicamos la secuencia:

```
AutoRemove(v, v)            { solo marcamos los nodos destino}
AutoShow(v, v)              { mostramos páginas y menús automáticamente }
Step(U, detalles, forward)  { seguimos la etiqueta etiquetada con detalles }
```

obtenemos el estado de usuario descrito en la Figura 2.12, donde se muestran concurrentemente los contenidos de las páginas 2, 4 y 5, y se muestran los menús asociados al nodo. Nótese como la descripción de la aplicación a través del hipergrafo afecta a la semántica de navegación. Es este hecho el que nos ha llevado a clasificarlo en la introducción como modelo hipermedia y no como modelo de referencia.

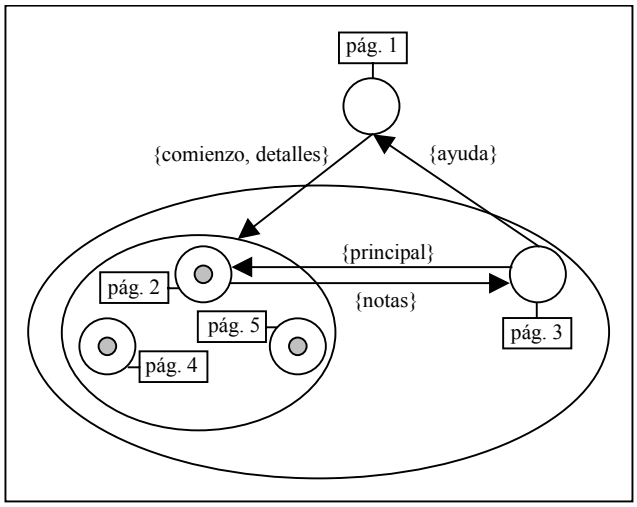

Figura 2.12 Nuevo estado de usuario

Si a partir de este estado aplicamos la secuencia:

AutoRemove(v, f) { solo eliminamos los marcadores de nodos origen}
AutoShow(v, f) { mostramos páginas; los menús bajo demanda }
Step(U, notas, forward) { seguimos la etiqueta etiquetada con notas }

obtenemos el estado descrito en la Figura 2.13, donde se muestra el contenido de las página 3, 4, y 5.

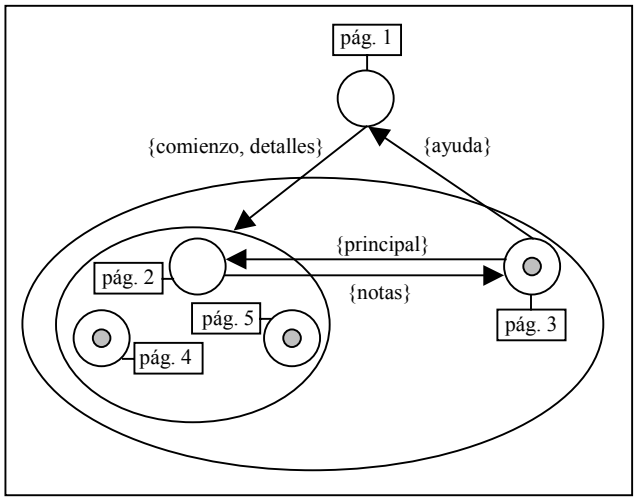

Figura 2.13 Nuevo estado de usuario

2.4.2.2 Ejemplo

Este modelo es incapaz de capturar la generación dinámica de contenidos, ya que esto conllevaría la generación dinámica de los nodos del hipergrafo que contienen a dichos contenidos, extremo no contemplado por el modelo.

Por otro lado la estructura presentacional del ejemplo muestra la mayor deficiencia de este modelo: la semántica de navegación no recae sobre la estructura de hipergrafo, sino sobre las primitivas de acceso al mismo, haciendo por tanto innecesaria la estructura del hipergrafo, e imposibilitando la posibilidad de proporcionar una semántica de navegación por defecto. En efecto, para representar el esquema navegacional descrito necesitamos un grafo como el mostrado en la Figura 2.14. Decimos grafo, porque como veremos en el ejemplo la estructura de hipergrafo no es necesaria para caracterizar la aplicación considerada. Nótese que la primera carencia del modelo surge al no permitir incluir la información sobre las diversas posiciones a las que se puede acceder a la **docencia** desde la **biografía**. Esto se debe a la falta de anclas en el modelo. Por otro lado, obviaremos la caracterización matemática del hipergrafo.

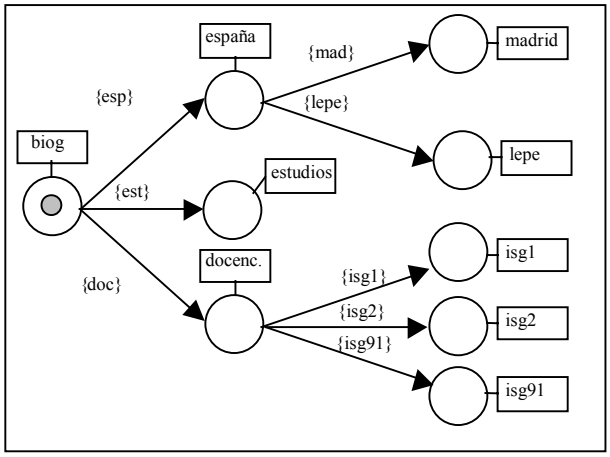

Figura 2.14 El hipergrafo que representa al ejemplo. Nótese como se está mostrando la biografía.

Si deseásemos seleccionar la foto de **españa**, tendríamos la siguiente secuencia de acciones (un guión _ indica que el valor es indiferente):

> Autoshow(true, true) { siempre vamos a mostrar el destino y sus menús }
> Autoremove(true, _) {queremos que desaparezca biografía, y no hay nodos finales}
> Step(U, esp, forward)

Situación descrita en la Figura 2.15.

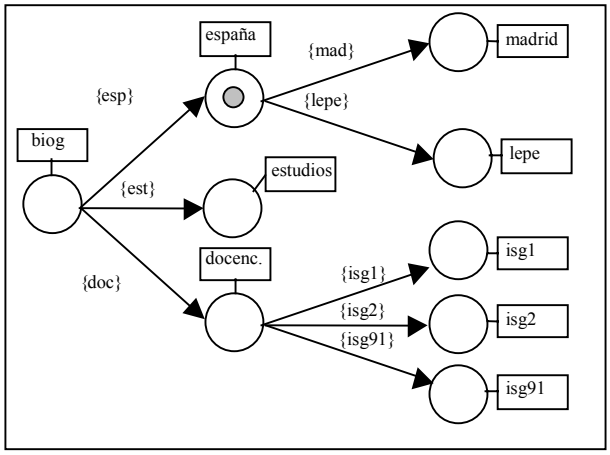

Figura 2.15 Situación actual del hipertexto.

Si ahora deseamos seleccionar el enlace con **madrid**, no tenemos más que aplicar:

> Autoremove(false, _) { no queremos que desaparezca españa y no hay nodos finales }
> Step(U, mad, forward)

Situación mostrada en la Figura 2.16.

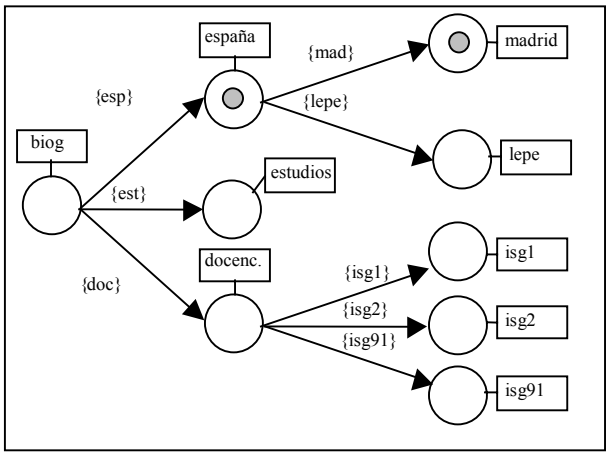

Figura 2.16 Situación actual del hipertexto.

Si ahora queremos activar el enlace entre la fotografía de **españa** y **lepe** no tenemos más que aplicar:

> Autoremove(false, true) {queremos mostrar españa, pero no madrid }
> Step(U, lepe, forward)

Estado representado en la Figura 2.17.

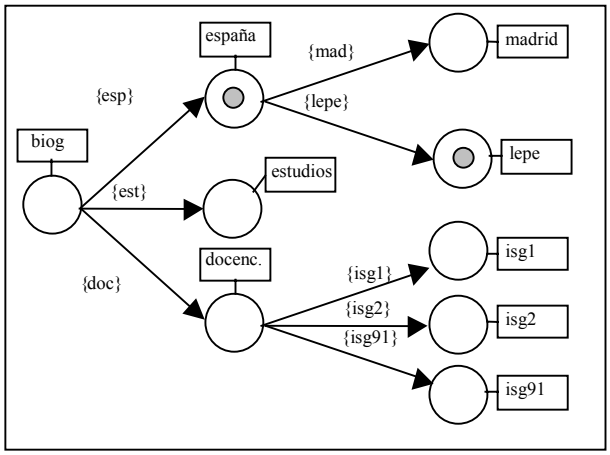

Figura 2.17 Situación actual del hipertexto.

Por último, si quisiéramos seleccionar el enlace de la fotografía de españa con la biografía, no tendríamos más que aplicar:

```
Autoremove (true, true) { debe desaparecer tanto españa, como lepe }
Step(U, esp, reverse)
```

Estado representado en la Figura 2.18.

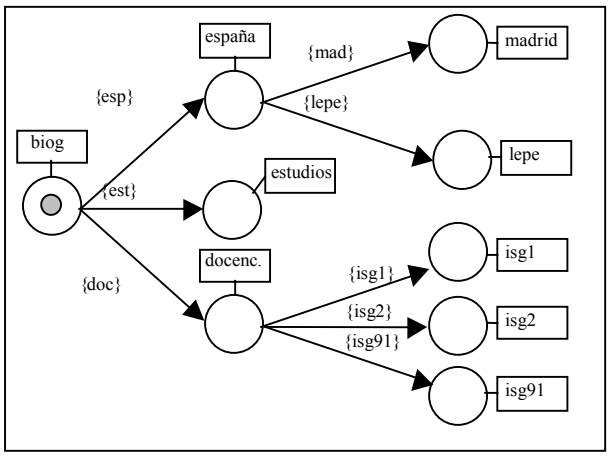

Figura 2.18 Situación actual del hipertexto.

Es decir, la mayor parte del peso de la semántica navegacional recae sobre las primitivas de presentación. De hecho, en este ejemplo la estructura de hipergrafo es totalmente innecesaria.

Por tanto la estructura de hipergrafo es un complemente a las primitivas de presentación para proporcionar semántica de navegación avanzada a la máquina abstracta de hipertexto. El modelo propuesto en esta tesis tiene una semántica de navegación similar a la del modelo Trellis. Una vez seleccionado un enlace (su ancla origen, mejor dicho) automáticamente, y sin necesidad de proporcionar explícitamente el código (o secuencia de funciones) se actualizará convenientemente el esquema navegacional. Esto presenta la ventaja de no tener que proporcionar código, junto con la posibilidad de contar con una semántica de navegación por defecto. Este último punto posibilitará la generación automática de prototipos basados en las descripciones proporcionadas por el modelo hipermedia [Nanard 98].

2.4.2.3 Discusión y comparativa

El modelo de hipergrafos representa una extensión del sistema HAM. Para capturar la noción de contexto utiliza la estructura de hipergrafo junto a varias primitivas de presentación, siendo superflua en algunas ocasiones la necesidad de dicho hipergrafo debido, precisamente, a la existencia de dichas primitivas.

Separación de contenidos, navegación y presentación. En este modelo los elementos del nivel de composición interna vienen representado por el concepto de página. Además incluye la noción de lectores como interpretes de dichas páginas. Ahora el nivel de almacenamiento sigue representado por un grafo (o por un hipergrafo), pero las posibilidades de presentación múltiple provienen de la noción de marcado de usuario, y de poder variarlo con las funciones Step y AutoRemove. El modelo obvia totalmente la representación de anclas como origen o destino de los enlaces. El nivel de ejecución viene marcado por las variaciones en el estado de usuario que se consiguen asociando las acciones en forma de funciones a una interfaz predeterminada.

Contexto. El modelo de hipergrafos es capaz de representar la noción de contexto, aunque como hemos visto, gracias a sus funciones de acceso. Además la representación explícita de todos los contenidos, y de todos los enlaces entre estos puede dificultar la comprensión del hipertexto.

Sincronización. El modelo no proporciona ninguna capacidad de sincronización. La razón puede deberse a su temprano origen (1989). Por otro lado la inclusión de más funciones que representasen estas capacidades podría ser una buena forma de incluir posibilidades de sincronización al modelo.

Formalización de la semántica de navegación. El modelo incluye un formalismo de grafo extendido para representar al hipertexto. Además la existencia de funciones bien definidas que caracterizan el estado del hipertexto se muestra idónea como manera de expresar la semántica de navegación del hipertexto. Estas ventajas, pueden convertirse en inconveniente si el tamaño de la aplicación es considerable, en cuyo caso se necesitarían de herramientas adecuadas para poder visionar la estructura del mismo. Nótese que la semántica de navegación viene determinada no por el hipergrafo, sino por las marcas que tenemos sobre los nodos, y por las funciones que actúan sobre estas marcas. Por tanto el modelo no proporciona semántica de navegación por defecto, lo que dificulta la construcción de entornos de desarrollo basados en el modelo.

Organización relacional de los contenidos. Al igual que todos los sistemas de representación analizados hasta el momento, y a su vez como la mayoría de los analizados en este capítulo, el modelo de hipergrafos no supone ninguna organización de los contenidos, y menos una organización relacional.

Gestión explícita de eventos. Al igual que la máquina abstracta de hipertexto, el modelo no incluye gestión de eventos.

Generación dinámica de contenidos. Como ya hemos comentado, este modelo es totalmente incapaz de caracterizar la generación dinámica de contenidos, ya que esto obligaría a la generación dinámica del propio hipergrafo, característica no considerada por los autores.

Gestión de usuarios y seguridad. Al ser un modelo que considera al hipertexto como una base de datos incluye una gestión explícita de usuarios a través del concepto de vista. Por otro lado, no incluye ninguna capacidad de seguridad.

Distinción de enlaces a nivel contenidos y esquema navegacional. El modelo no distingue entre dos niveles de enlaces, estando confiada la estructura hipertextual al nivel de almacenamiento, es decir, al hipergrafo.

Técnicas orientadas a objetos. Al igual que HAM no entra dentro de los intereses del modelo de hipergrafos ninguna característica ajena a representar la estructura navegacional del hipertexto, o la semántica de navegación que se deriva exclusivamente de esta estructura.

2.4.3 El modelo Trellis

2.4.3.1 Presentación

Introducción

El modelo Trellis [Stotts 89], al igual que el de hipergrafos, puede considerarse como una extensión a la máquina abstracta de hipertexto con el fin de representar la capacidad de mostrar varios contenidos simultáneamente, los cuales se actualizan independientemente. Para lograr este objetivo, el modelo va a representar la capa de almacenamiento, y parte del nivel de ejecución, con una red de Petri. Ahora, va a ser la propia red la encargada de modelar la representación simultánea de varios contenidos, y la semántica de navegación. Al igual que el modelo Amsterdam va a incluir un sistema integrado capaz de pasar de la representación de un hipergrafo representado a través de una red de Petri, a una aplicación hipertexto ejecutable. A diferencia del modelo de hipergrafos no necesitará funciones auxiliares para codificar la semántica de navegación del hipertexto, estando esta codificada en la red de Petri.

Teoría básica de redes de Petri

A continuación dedicaremos unas líneas a describir la teoría básica de redes de Petri. Una *estructura de red de Petri* es una terna N = <S, T, F>, donde

- $S = \{s_1, ..., s_n\}$ es un conjunto finito de *lugares* con $n \geq 0$;
- $T = \{t_1, ..., t_m\}$ es un conjunto finito de *transiciones* con $m \geq 0$ y $S \cap T = \varnothing$;
- $F \subseteq (S \times T) \cup (T \times S)$ es la *relación de flujo*, una aplicación que representa arcos entre lugares y transiciones.

Los arcos representados por F describen pre- y post-relaciones para lugares y transiciones. El conjunto de lugares que inciden en una transición t se denomina el *preconjunto* (del inglés *preset*) de t, y se representa por:

$$\bullet t = \{ s \mid (s, t) \in F \}$$

El conjunto de lugares que siguen a una transición se denomina el *postconjunto* (del inglés *postset*) de t, y se representa por:

$$t\bullet = \{ s \mid (t, s) \in F \}$$

El preconjunto y el postconjunto para un lugar s se definen similarmente como los conjuntos de transiciones incidentes sobre s y que se siguen de s, respectivamente.

En el modelo Trellis se simplifica la notación normalmente utilizada para redes de Petri, asumiendo que el peso de cada arco es uno, y que la capacidad de marcas (del inglés *token*) de cada lugar es ilimitada.

Un *marcado M*, de una estructura de red de Petri N = <S, T, F> es una función

$$M: S \rightarrow \{0, 1, ... \} \cup \{\omega\}$$

que asocia cada lugar en la red a un entero no negativo o al símbolo ω. Un marcado se representa normalmente como un vector $(m_1, m_2, ..., m_n)$ donde $m_i = M(s_i)$. Cada entero en un marcado indica el número de marcas que residen en el lugar correspondiente. El símbolo ω representa una cantidad arbitraria de tokens, y se incluye en el modelo por consistencia con la red de Petri no por su utilidad.

Una *ejecución* de una red de Petri consiste de una secuencia de marcados, empezando con un marcado inicial M_0, y terminando en un marcado final M_f. Para pasar del estado actual M al estado siguiente M', se elige cualquier transición t activa bajo M y se dispara. Decimos que una transición t está activa bajo M, sii

$$\forall\ s \in \bullet t, M(s) \geq 1$$

Disparar una transición activa consiste en eliminar una marca de cada lugar •t, y añadir un toquen a cada lugar en t•. El nuevo estado M' es la tupla de enteros que representa el número de marcas en cada lugar después del disparo.

Hipertexto

Según el modelo Trellis, un *hipertexto* es una tupla $H = <N, C, W, B, P_l, P_d>$, donde

- N = <S, T, F> es una estructura de red de Petri.

- C es un conjunto de *documentos de contenido*.
- W es un conjunto de *ventanas*.
- B es un conjunto de *botones*.
- P_l es una *proyección lógica* para el documento.
- P_d es una *proyección de presentación* para el documento.

La red de Petri se va a utilizar para representar el nivel de almacenamiento en el modelo. Ahora este nivel va a contener la estructura de enlaces entre diversos componentes (los contenidos, ventanas y botones), y dos funciones llamadas proyecciones, una entre la red de Petri y los componentes, y otra entre la red de Petri y los mecanismos de presentación. Una ventana del conjunto W es un lugar distinguido de información, que puede contener texto, imagen, vídeo, etc. (es decir, es un elemento similar al de canal en el modelo Amsterdam). Un botón del conjunto B es una acción que causa el cambio de presentación en un modo predeterminado (similares como veremos a los *activadores de nexos* del modelo Pipe). Un elemento de contenido del conjunto C puede ser diversas cosas: texto, gráficos, código ejecutable, o incluso otro hipertexto. La Figura 2.19 contiene un hipertexto representado a través de una red de Petri.

Dada la estructura de red de Petri N = <S, T, F> en un hipertexto H, la *proyección lógica* de H es una terna P_l = <C_l, W_l, B_l> donde,

- C_l: $S \rightarrow C \cup \{\nu\}$
- W_l: $S \rightarrow W \cup \{\nu\}$
- B_l: $T \rightarrow B \cup \{\nu\}$

El elemento ν representa un valor nulo. La función C_l asocia un elemento de contenido a cada lugar de la red de Petri. La función W_l asocia una ventana lógica con cada lugar de la red. La función B_l asocia un botón lógico con cada transición. Una transición se dispara seleccionando su botón lógico. Los botones por tanto son un mecanismo que asocia la presentación del hipertexto con una ejecución de la red de Petri. Estas aplicaciones proporcionan la información y el esquema lógico para la presentación de la red de Petri.

La *proyección de presentación* P_d es una colección de aplicaciones que asocian los botones y ventanas lógicas de un hipertexto con representaciones físicas en pantallas y localizaciones. Por ejemplo, dado un segmento de texto mostrado en una ventana física, un modo de representar los botones lógicos es asignarlos un menú seleccionable en otra ventana (esta es la aproximación del entorno de diseño αTrellis). Otro mecanismo sería asociarlos a palabras de la ventana de texto.

Un *hipertexto marcado* es un par H_M = <H, M> donde H = <N, C, W, B, P_l, P_d> es un hipertexto, y M es un marcado de la red de Petri N en H. El hipertexto marcado es la forma elegida por el modelo para representar el estado de un hipertexto durante su presentación. Un caso especial de hipertexto marcado, denominado *estado inicial*, es H_{M0} = <H, M_0> donde M_0 es un marcado inicial para la red de Petri en H. Cuando un hipertexto es visionado por primera vez, los contenidos de los nodos que se muestran son los correspondientes a los lugares que contienen marcas en el marcado inicial M_0, es decir, el conjunto de elementos mostrados es:

$$\{C_l(s) \mid s \in S \wedge M_0(s) > 0\}$$

La semántica de ejecución de la red de Petri proporciona el modelo de presentación para un hipertexto marcado. Una marca en un lugar s indica que los contenidos del lugar $C_l(s)$ se muestran. Cuando una marca se mueve a un lugar vacío, el elemento de contenido asociado se asocia a un dispositivo de presentación. Por el contrario, cuando todos los lugares se eliminan de un lugar, dejándolo vacío, los contenidos de ese lugar dejan de mostrarse al usuario. Las marcas se mueven a través de la red de Petri cuando son disparadas. Esto se logra seleccionando los botones lógicos en la presentación. Cuando una transición t se activa en la red de Petri, el botón lógico $B_l(t)$ se asocia a algún área en la pantalla donde pueda seleccionarse. La presentación comienza con la ejecución de la red de Petri en M_0, y puede terminar o ciclar, en función de la estructura del hipertexto. Si se llega a un estado M sin ninguna transición activa, la presentación termina, ya que no hay botones seleccionables en ese punto. Cuando se llega a un estado que contiene un hipertexto anidado se activa el estado inicial de ese hipertexto. Saldremos de él, cuando no se tenga activa ninguna transición, o cuando se seleccione una transición fuera de él.

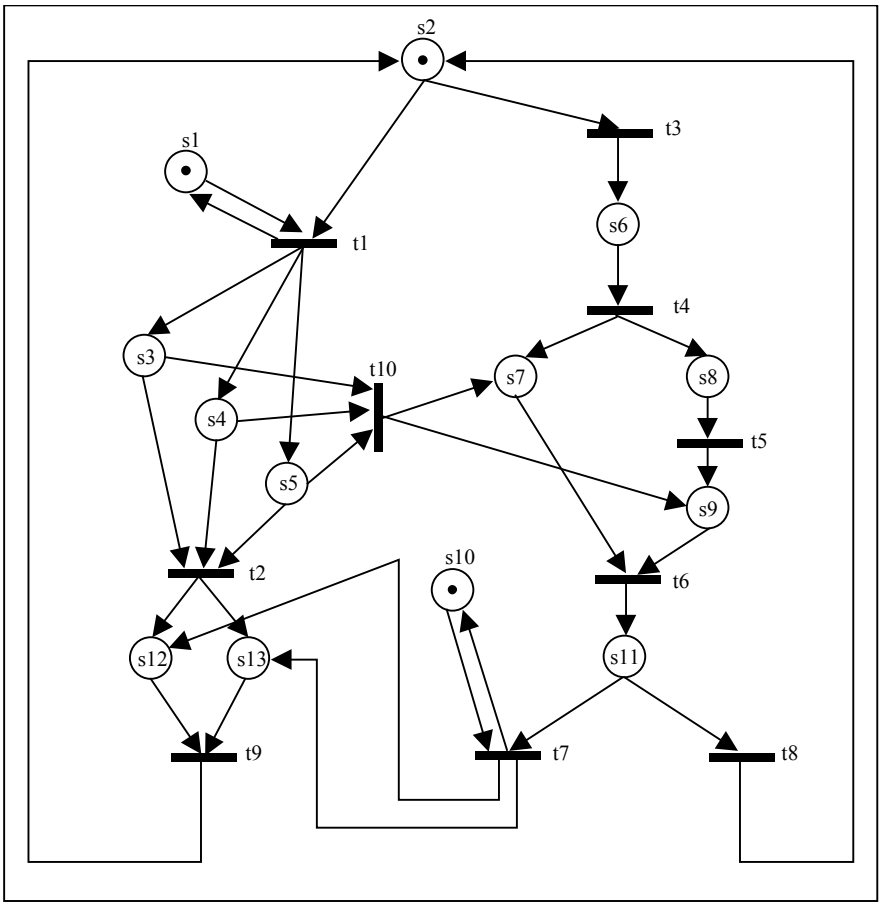

Figura 2.19 Representación de un hipertexto mediante una red de Petri. Las marcas indican que se está mostrando el contenido asociado a los lugares s1, s2 y s10, pudiendo ser seleccionados los botones asociados a las transiciones t1 y t3.

El sistema de hipertexto αTrellis

El modelo Trellis incluye un sistema de hipertexto, capaz de generar aplicaciones hipermedia (dentro del entorno del propio sistema) a partir de la representación proporcionada por la red de Petri. Este sistema permite la visión del hipertexto como una red de Petri. A cada estado marcado se le hace corresponder una ventana de la aplicación, y a cada transición de la red se le asigna una palabra seleccionable en un lateral de la ventana. Es decir, la función W_l asocia una ventana a cada nodo, y la función B_l asocia una especie de *botonera* a las transiciones seleccionables. La función C_l asocia contenidos en base al nombre del archivo de los mismos. Este sistema presenta la ventaja de proporcionar aplicaciones hipertextuales en base a la representación abstracta del hipertexto, pero también presenta dos graves inconvenientes: es necesario la existencia de un sistema propietario para ejecutar la aplicación hipermedia, y no se puede incluir en la misma ninguna característica que no soporte el sistema.

Ventajas proporcionadas por el modelo

El modelo presenta diversas ventajas que se derivan directamente del uso del sistema de representación del hipertexto. Veamos cuales son estas ventajas:

- Utilizando la red de Petri, y el grafo de alcance derivado de esta, se puede determinar la complejidad de la presentación, como por ejemplo, el número de ventanas máximo que se va a mostrar simultáneamente.
- La red de Petri permite la visualización de la representación concurrente de los diversos contenidos de una aplicación hipermedia.
- La red de Petri puede usarse para determinar si durante la presentación de la aplicación hay partes del hipertexto que serán visualizadas, o que por el contrario, y en base al marcado inicial, jamás serán alcanzables.

- Utilizando arcos etiquetados se puede garantizar control de acceso a los usuarios. De esta forma, y a partir de un hipertexto, se pueden asociar distintas hipertextos marcados a distintos usuarios.
- Utilizando distintos marcados iniciales se pueden obtener versiones específicas de un hipertexto, el cual puede variar de una sesión a otra. Además variando la función C_l se pueden asignar distintos contenidos al mismo esquema navegacional.

2.4.3.2 Ejemplo

El modelo Trellis es incapaz de capturar la noción de contenido generado dinámicamente. Por lo tanto nos restringiremos a la parte estática. En cuanto a la interfaz el modelo no es capaz de representarla sin problemas. En efecto, el ejemplo queda descrito por la red de Petri de la Figura 2.20. Nótese que al igual que en el ejemplo del hipergrafo no podemos representar los múltiples enlaces que existen entre la biografía y la docencia.

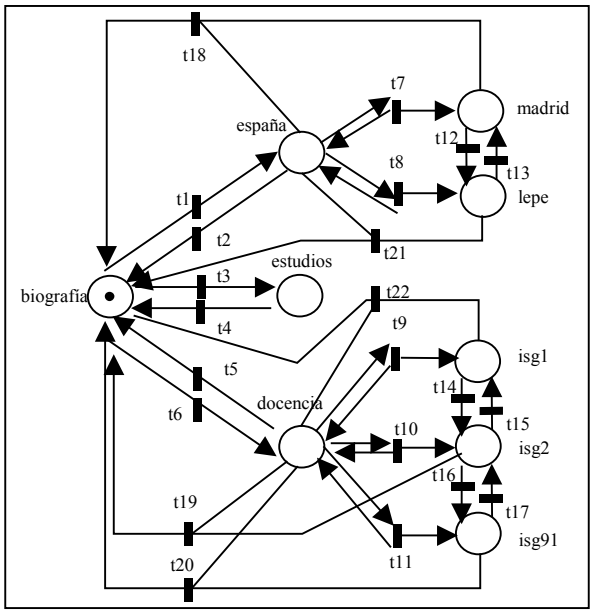

Figura 2.20 Representación del hipertexto a través de una red de Petri. Hemos incluido los contenidos asociados a los lugares para facilitar la legibilidad.

En esta figura podemos observar el marcado inicial correspondiente a la presentación de la biografía. Cuando se dispara la transición t1 se muestra el contenido de españa, tal y como ilustra la figura 2.21.

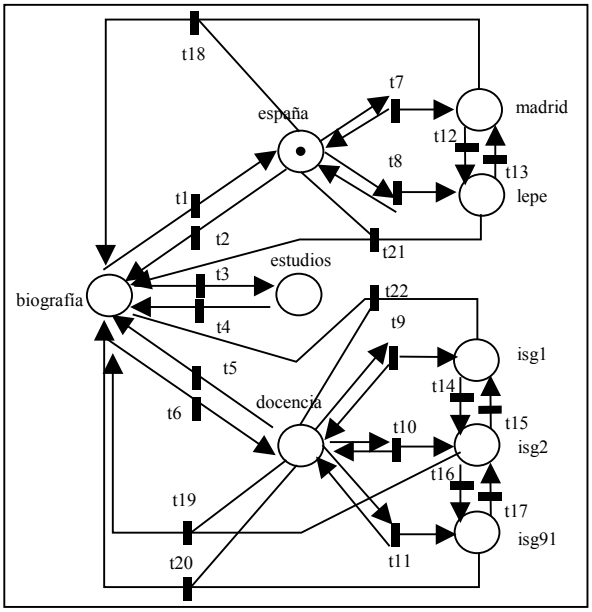

Figura 2.21 Estado del hipertexto tras la activación de la transición t1.

Cuando se dispara la transición t7 se muestra el contenido de madrid, tal y como ilustra la Figura 2.22.

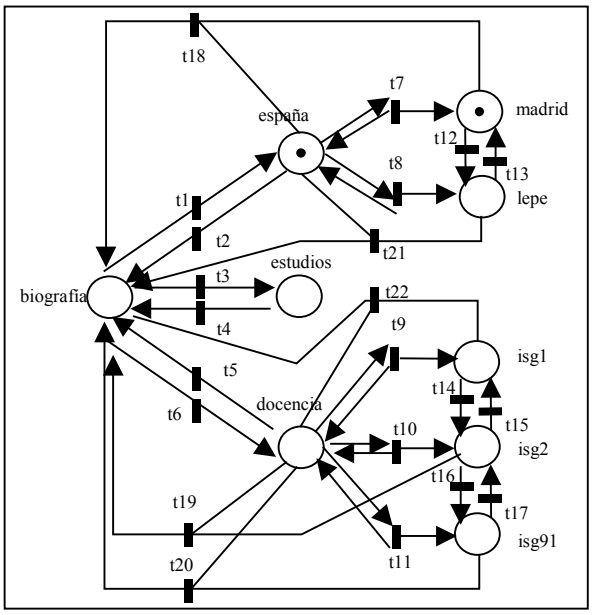

Figura 2.22 Estado del hipertexto tras la activación de la transición t7.

Si se desea volver a la biografía debería dispararse la transición t18. La Figura 2.23 ilustra el resultado.

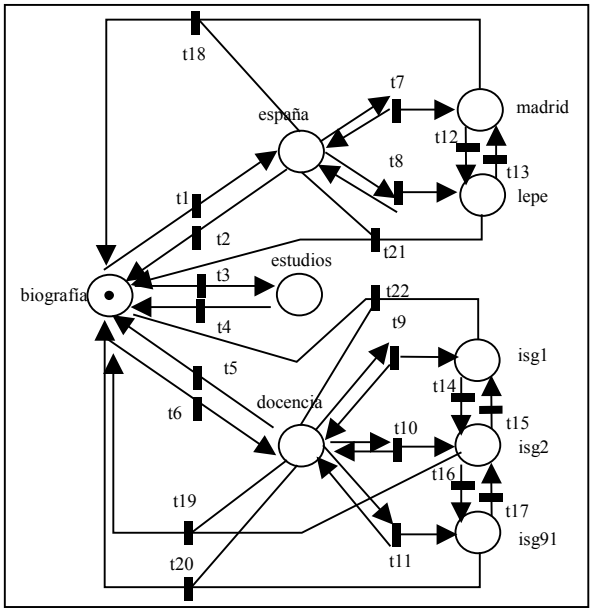

Figura 2.23 Estado del hipertexto tras la activación de la transición t18.

El problema radica en que en el hipertexto de la Figura 2.22 también podríamos haber activado la transición t2, en cuyo caso accederíamos a la biografía, pero no dejaríamos de mostrar madrid. La Figura 2.24 ilustra este supuesto. Este problema es de difícil solución, ya que necesitamos una transición que nos permita acceder de españa a biografía (t2), pero también otra que nos permita acceder desde españa y madrid a biografía (t18). Si quitamos t2, no tenemos acceso sin pasar por madrid. Si quitamos t18, no removemos madrid al acceder a biografía.

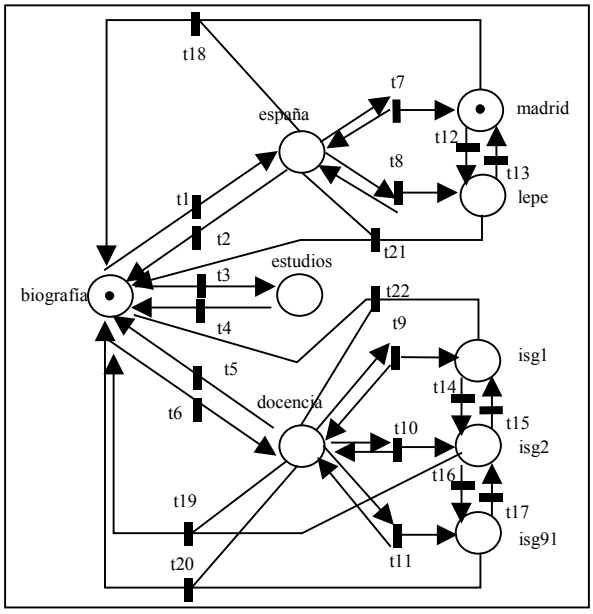

Figura 2.24 Estado del hipertexto si en la Figura 2.22 hubiésemos activado la transición t2.

2.4.3.3 Discusión y comparativa

El modelo Trellis es una ampliación al modelo propuesto por la máquina abstracta de hipertexto. De forma similar al modelo de hipergrafos permite representar la visualización simultánea de distintos contenidos, recurriendo en esta ocasión a una red de Petri. El problema es que esta estructura presenta problemas a la hora de caracterizar las aplicaciones hipermedia que se diseñan en la actualidad, como es el caso del ejemplo anterior.

Separación de contenidos, navegación y presentación. En este modelo hay una clara separación entre el nivel de composición interna y el de almacenamiento. Ahora el nivel de almacenamiento viene representado por una red de Petri, estructura mucho más potente que la de grafo. La relación entre ambos niveles no se produce a través del concepto de ancla, sino a través de la función C_l. Ahora las anclas vienen representadas por a función B_l. El nivel de presentación viene representado por la función W_l y por la propia semántica de navegación asociada a la Red de Petri.

Contexto. Al igual que el modelo de hipergrafos, el modelo Trellis fue de los primeros en representar la idea de contexto. Aunque una red de Petri es un formalismo ideal para representar procesos concurrentes, vemos como la noción de enlace enturbia esta facilidad de representación. De esta forma vemos como en la Figura 2.20 la red de Petri presenta importantes problemas a la hora de representar el esquema navegacional del ejemplo.

Sincronización. Es posible utilizar una red de Petri temporal asociando un entero a cada lugar de la red de Petri. Este entero representa el número de unidades de tiempo que deben transcurrir antes de que en el lugar se puedan activar sus transiciones de salida. En términos del hipertexto, el entero indica la mínima cantidad de tiempo que los contenidos asociados a un lugar van a ser mostrados en pantalla. Este es un soporte limitado de la sincronización, muy lejos del proporcionado por el modelo Amsterdam.

Formalización de la semántica de navegación. La red de Petri es el formalismo utilizado para representar el nivel de almacenamiento y el esquema navegacional. Aunque en principio parece una buena idea utilizar este mecanismo para el esquema navegacional de las aplicaciones hipermedia, vemos en el ejemplo que el concepto de contexto, unido a un posible incremento de los contenidos de la aplicación puede dificultar bastante el uso de una red de Petri para representar un hipertexto de complejidad media. Cuando se encuentra un marcador sobre un estado, se muestran los contenidos asociados a ese marcador. Las transiciones se traducen por botones, y seleccionar un botón es equivalente a activar una transición. En cualquier caso el contar con una semántica de navegación por defecto posibilita el desarrollo de entornos que

generan automáticamente prototipos/aplicaciones en base a las descripciones proporcionadas por el modelo (Capítulo 3). Esta aproximación será la seguida por el modelo hipermedia presentado en el Capítulo 5, y por el modelo de proceso presentado en el Capítulo 6.

Organización relacional de los contenidos. Al igual que los sistema de representación precedentes el modelo Trellis no supone ninguna propiedad sobre los contenidos. Tampoco especifica como asignar anclas a los mismos. En el sistema αTrellis este problema se solventa mediante el concepto de botonera, asignando a cada transición un lugar en la misma.

Gestión explícita de eventos. Este modelo no soporta ninguna gestión explícita de eventos. Sí considera, sin embargo, la posibilidad de asociar programas ejecutables como contenidos de los nodos, característica no presentada, en principio, de manera explícita por ningún otro sistema de representación hipermedia. Esta característica quedará recogida en el modelo propuesto en esta tesis a través del concepto de proceso subordinado.

Generación dinámica de contenidos. Como ya hemos comentado, el modelo no contempla la posibilidad de caracterizar contenidos dinámicamente generados, ya que esto conllevaría la generación dinámica de la propia red de Petri.

Gestión de usuarios y seguridad. El modelo permite dar cabida a distintos usuarios a través del concepto de marcado. En función del marcado inicial, distintos usuarios podrán tener distintas capacidades de visionado del hipertexto. Además incluye la noción de red de Petri coloreada para aumentar la potencia de esta aproximación. La seguridad está limitada a las posibilidades que permiten los mecanismos previamente descritos.

Distinción de enlaces a nivel contenidos y esquema navegacional. El modelo no distingue entre enlaces a nivel de contenidos y a nivel esquema navegacional. Esto obliga, al igual que en los sistemas de representación anteriores, a representar todos los enlaces como relaciones de flujo asociadas a todos los lugares, complicando sobremanera la red de Petri.

Técnicas orientadas a objetos. Aunque, como ya hemos comentado, el modelo permite asociar programas ejecutables como contenidos de los nodos, este no concibe la utilización de técnicas orientadas a objetos. Al igual que el modelo de hipergrafos, la fecha de origen de este modelo (1989) tampoco facilita esta posibilidad.

2.5 Sistemas de representación relacionales

Hasta ahora los sistemas de representación considerados no suponían ninguna organización específica de los contenidos más allá de la meramente hipermedia. Los modelos HDM, y RMDM suponen una organización relacional de los mismos que va a ser determinante para determinar a posteriori el esquema navegacional de la aplicación. La razón estriba a que la estructuración de los contenidos va a ser tan fuerte que difícilmente se va a poder obviar a la hora de especificar los enlaces entre estos. Muy adecuados para proporcionar una visión hipertextual de bases de datos, estos modelos adolecen de capacidades para representar ideas tan fundamentales en hipermedia como la de contexto o sincronización. Como los propios autores de RMDM reconocen [Isakowitz 95], este tipo de modelos tampoco se ajustan bien a cualquier aplicación cuyos contenidos no puedan organizarse a través de un esquema Entidad-Relación [Chen 76]. De hecho, autores como [Balasubramanian 97] en diversas ocasiones han optado por no utilizar HDM, RMDM, y ni siquiera OOHDM, en el diseño de un sistema a gran escala debido a la organización relacional que estos imponen.

2.5.1 El modelo de diseño de hipertextos (HDM)

2.5.1.1 Presentación

Introducción

El Modelo de Diseño de Hipertextos, HDM (*Hipertext Design Model*) [Garzotto 93], surge como un modelo de *autoría a gran escala* (del inglés *authoring-in-the-large*), es decir un modelo que se ocupa de la especificación y diseño de las características globales y estructurales de un hipertexto. En contraposición, el modelo no se

preocupa de la *autoría en detalle* (del inglés *authoring-in-the-small*), centrado en el desarrollo de los contenidos de la aplicación. Nótese que esta aproximación es común a todos los sistemas de representación hipermedia de este capítulo, así como para el modelo presentado en el Capítulo 5. Con este fin HDM utiliza la potencia proporcionada por el modelo relacional para representar las aplicaciones hipertextuales, aumentándolo convenientemente para soportar el concepto de hiperenlace. Además este modelo puede ser utilizado para proporcionar la entrada de un sistema hipertexto (similar al modelo αTrellis) que genere la aplicación hipermedia. Esto obligará a una definición explícita del concepto de *semántica de navegación por defecto* para el modelo.

Una aplicación HDM esta formada por estructuras que contienen información denominadas *entidades*. Una entidad denota un objeto conceptual o físico del dominio. Las entidades se agrupan en *tipos de entidades*. De esta forma podemos tener el tipo de entidad CIUDAD, y como entidades concretas Madrid o Tokio. Una entidad es la menor pieza autónoma de información que puede ser insertada o eliminada de una aplicación. En este contexto, autónoma significa que su existencia no está condicionada por la existencia de otros objetos de información. En HDM, solamente las entidades son autónomas, en contraposición a los componentes y unidades.

Una entidad es una jerarquía de *componentes*, estando formados estos por *unidades*. Cada unidad muestra el contenido de una componente bajo una determinada *perspectiva* (por ejemplo, contenidos bajo la perspectiva inglés, o bajo la perspectiva español). Por tanto las entidades derivan sus contenidos de sus componentes, y estos a su vez de sus unidades. Las unidades son la menor pieza de información que puede ser visualizada en un hipertexto.

Las estructuras de información HDM pueden interconectarse a través de *enlaces*. HDM distingue entre tres categorías de enlaces. Los *enlaces estructurales* conectan los componentes que pertenecen a una misma unidad. Los *enlaces de perspectiva* conectan entre sí a las unidades que pertenecen al mismo componente. Finalmente los *enlaces de la aplicación* denotan relaciones arbitrarias, dependientes del dominio que conectan componentes y entidades, del mismo o distinto tipo. Es decir, los enlaces de la aplicación representan los enlaces hipermedia de la aplicación. Los enlaces de la aplicación se agrupan en *tipos de enlaces*, los cuales se establecen entre tipos de entidades.

Como todo modelo relacional, HDM hace una clara distinción entre la noción de *esquema* y de *instancia de esquema*. Un esquema es una colección de definiciones de tipos que describen una aplicación a nivel global. Una instancia de un esquema es una colección de entidades, componentes, unidades, y enlaces que satisfacen las definiciones del esquema. Los *perfiles* (del inglés *outline*) son estructuras de acceso que proporcionan puntos de entrada para los usuarios a las estructuras de información en una instancia de un esquema.

Una *semántica de navegación* tiene el propósito de especificar como se van a mostrar al lector las estructuras de información, y como puede navegar por ellas. HDM proporciona una *semántica de navegación por defecto*, pudiéndose especificar otras para obtener mayor potencia de visualización. Sin embargo, el modelo no determina como especificar estas otras semánticas de presentación.

Finalmente hay evoluciones posteriores del modelo como [Garzotto 95] que aumentan las posibilidades de presentación del mismo, pero que no lo varían sustancialmente.

Entidades y tipos de entidades

Una *entidad* es una estructura de información que representa algún objeto del dominio de la aplicación; una ley (digamos la Ley 19/8/89), una ópera musical (digamos La Traviata, de Verdi), o una pieza mecánica (digamos Motor eléctrico) pueden ser ejemplos de entidades. Las entidades se agrupan en *tipos de entidades* que se corresponden con clases de objetos del mundo real. LEY, OPERA MUSICAL, y PIEZAS son ejemplos de tipos de entidades.

La noción de entidad y de tipo de entidad es una noción adecuada para modelar información en el mundo de las bases de datos. El modelo de diseño más popular en este ámbito, el modelo *Entidad-Relación*, E-R [Chen 76], y otros derivados de este, se basan en la noción de entidad y de tipo de entidad. Sin embargo, las entidades HDM difieren sustancialmente de las entidades del modelo E-R. Por ejemplo, las entidades HDM tienen complejas estructuras internas (con enlaces internos), mientras que las entidades E-R son básicamente planas; las entidades HDM tienen una semántica de navegación por defecto asociada; las entidades HDM

pueden ser interrelacionadas a través de enlaces de aplicación, en patrones mucho más complejos que los permitidos por las relaciones del modelo E-R.

Componentes

Una entidad HDM es una colección de componentes organizados en una jerarquía arbórea. Un *componente* es una abstracción para un conjunto de unidades, que son los contenedores reales de la información, y que derivan su contenido de sus unidades. Un componente que forma parte de una jerarquía, por lo general tiene un *padre* (salvo el componente raíz), un número de *parientes* (que se organizan en orden lineal), y un número de *hijos* (salvo para los componentes hoja). Los componentes heredan su tipo de su entidad y solo pueden existir como parte de la misma. En este sentido, los componentes no son autónomos.

Como ejemplo de componentes para las entidades introducidas en el apartado anterior, podemos tener Artículo 1 (componente de la entidad Ley 19/8/89), Artículo1-Subsección 1.2 (hijo del componente Artículo 1), Obertura (componente de La Traviata), o condensador (componente de Motor eléctrico).

Perspectivas

En las aplicaciones hipertextuales es corriente tener que presentar la misma información desde distintos puntos de vista. El idioma, el tipo de medio audiovisual, o la dificultad para el usuario, pueden ser distintos puntos de vista. HDM representa la noción de punto de vista a través del concepto de *perspectiva*. Si una entidad tiene dos perspectivas posibles, (por ejemplo Español e Inglés, o Audio y Vídeo), todos los componentes pertenecientes a la entidad tendrán ambas perspectivas (esto quiere decir, por ejemplo, que las unidades que forman los componentes que determinan a los componen de las OPERAS deberán estar en formato audio, y en formato vídeo). Las perspectivas son comunes a las entidades del mismo tipo, y se definen al nivel de tipo de entidad.

Unidades

Una *unidad* es el contenido que le corresponde a un componente bajo una determinada perspectiva. Una unidad se caracteriza por un *nombre* (su identificador) y un *cuerpo*. Los cuerpos de las unidades son el contenido real de una aplicación HDM. Ejemplos de unidades pueden ser: Obertura/imagen, Obertura/vídeo (de la entidad La Traviata); Artículo 1/texto oficial, Artículo 1/texto anotado (de la entidad Ley 19/8/89); instrucciones de montaje/texto español, instrucciones de montaje/texto inglés, (de la entidad Motor eléctrico).

Proporcionar el cuerpo de una unidad se corresponde con la autoría en detalle, quedando fuera de los objetivos del modelo HDM. Además se permite que distintas unidades compartan el mismo cuerpo. Aunque no está recomendado, esto permite que el mismo contenido sea accedido desde ámbitos distintos. La Figura 2.25 ilustra estos conceptos. En este ejemplo tenemos la entidad ejercicio 8 formado por los componentes enunciado y solución. En el ejemplo solo hay un nivel en la jerarquía de componentes, pero esto no tiene por que ser así. La entidad tiene dos perspectivas (español e inglés), y cuatro unidades, una por cada componente y perspectiva.

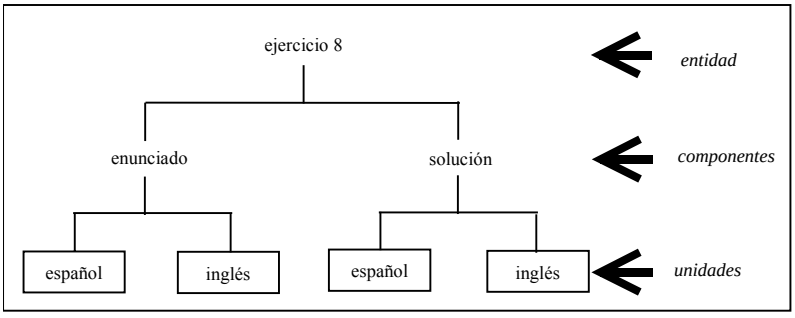

Figura 2.25 Entidades, componentes y unidades

Enlaces

HDM considera los enlaces con dos facetas: representacional y navegacional. En la primera, los enlaces sirven para capturar las relaciones de dominio, mientras que en la segunda su misión es capturar los patrones de

navegación existentes en la aplicación. Entre los enlaces representacionales tenemos los enlaces de perspectiva y estructurales. Entre los navegacionales tenemos los enlaces de aplicación. Con independencia del tipo de enlace que se defina, al final todos servirán para presentar las entidades y componentes de una manera hipertextual.

Los *enlaces de perspectiva* interconectan unidades correspondientes al mismo componente. Los enlaces de perspectiva conectarían las unidades enunciado/español y enunciado/inglés para conmutar de una descripción de un idioma a otro. Desde el punto de vista navegacional, estos enlaces son muy sencillos para el lector, ya que al activarlo no se varía la entidad sobre la que se centra el lector.

Los *enlaces estructurales* conectan componentes pertenecientes a la misma entidad. Hay distintos enlaces estructurales, correspondientes a la relación inducida por la estructura de árbol ordenado de las entidades. Ejemplos de enlaces estructurales son Up conectando un componente a su padre, Down, conectando un componente a su hijo, Down-1, conectando un componente a su primer hijo, Next-sibling, conectando un componente al siguiente hermano del mismo padre, Root, conectando un componente con la raíz del árbol, y así. Los enlaces estructurales pueden ser utilizados por el lector para navegar entre las piezas de información que componen una entidad. Navegacionalmente, son un poco más complejos que los enlaces de perspectiva, pero siguen siendo simples, ya que dejan al lector dentro de la misma entidad. Además en el peor de los casos, el enlace Root siempre puede llevar al lector a la raíz de la entidad. En la práctica los enlaces estructurales se suprimen, tomando la entidad como un todo formado por los componentes especificados por los enlaces estructurales.

Los *enlaces de aplicación* representan las relaciones del dominio existentes entre entidades, o sus componentes. Los enlaces de aplicación se organizan en tipos. Un *tipo de enlace de aplicación*, o *tipo de enlace*, se especifica mediante un *nombre*, un conjunto de *tipos de entidad origen* y *destino*, y un *atributo de simetría*, que puede tomar los valores *simétrico* o *asimétrico*. Los tipos de entidad origen y destino definen enlaces entre entidades de estos tipos. De esta forma cuando definimos un tipo de enlace con origen en el tipo de entidad A, y con destino en el tipo de entidad B, se permiten *instancias* de este tipo de enlaces entre entidades de tipo A y entidades de tipo B. El atributo de simetría indica cuando el enlace es en ambas direcciones. Un ejemplo de enlace de aplicación es autor-de, cuyas instancias conectan entidades o componentes del tipo Compositor a entidades o componentes del tipo Ópera Musical. Desde un punto de vista navegacional estos enlaces son los más complejos ya que cambian la entidad que está siendo presentada al lector.

Si ahora revisamos el concepto de *tipo de entidad* vemos que todas las entidades pertenecientes al mismo tipo tienen ciertas características en común: el nombre de su tipo de entidad, el conjunto de perspectivas bajo las que se puede presentar sus contenidos (es decir, el cuerpo de sus unidades), y los tipos de sus enlaces de aplicación origen y destino.

Esquema HDM

Una especificación HDM de un hipertexto está formada por una *definición de esquema* y un conjunto de *definiciones de instancias*. Una definición de esquema especifica que conjunto de tipos de entidades y de tipos de enlaces. De esta forma solo se pueden insertar instancias en la aplicación si satisfacen las restricciones impuestas por los esquemas.

Perfiles

Según la terminología HDM, una aplicación hipermedia puede dividirse en dos partes: una *hiperbase* y un conjunto de *estructuras de acceso*. La hiperbase representa el núcleo de la aplicación y está formada por entidades, componentes, unidades, y enlaces (estructurales, de perspectiva, y de aplicación). El lector puede explorar la hiperbase atravesando los enlaces allí definidos.

Sin embargo, antes de empezar esta navegación, el usuario debe tener puntos de entrada para tomar conciencia de los contenidos de la hiperbase, y localizar el punto de entrada más conveniente. Las estructuras de acceso tienen el propósito de permitir al lector el elegir los puntos de acceso para comenzar la navegación. Con este fin HDM define los *perfiles*. Un perfil es un árbol ordenado de componentes. Para seleccionar un componente u otro se pueden utilizar los enlaces estructurales entre estos. Para navegar a una entidad o componente de la hiperbase, se debe seleccionar un enlace de los componentes hoja.

Enlaces derivados y tipos de enlaces derivados

HDM permite que los enlaces se definan de forma implícita (inducidos de propiedades estructurales), o pueden ser definidos intensionalmente y derivados de forma algorítmica. Todos los enlaces de perspectiva se pueden dejar implícitos y se derivan automáticamente de la definición de componente y unidades. Por ejemplo, si para el tipo de entidad Ópera, tenemos definidas las perspectivas audio y vídeo, se generan dos enlaces de perspectiva para cada componente de una entidad de este tipo. El primero conectará la unidad del componente bajo la perspectiva de audio a la unidad del componente bajo la perspectiva de vídeo. El segundo es el inverso del primero.

Definir una entidad es equivalente a definir un conjunto de componentes más el conjunto mínimo de enlaces estructurales que son necesarios para inducir una relación jerárquica ordenada sobre el conjunto de componentes: enlaces de un componente (no hoja) a su primer hijo, y enlaces de un componente a sus hermanos. Un gran número de enlaces estructurales pueden definirse intensionalmente, y pueden ser calculados a partir de los básicos. Por ejemplo down(3), conectando un componente a su tercer hijo, down, conectando un componente a todos sus hijos, etc. De una forma similar también se pueden derivar enlaces de aplicación. Así si dos entidades están conectadas por un enlace entre dos componentes, se puede derivar un enlace que conecte las raíces de ambas entidades a partir de este. Es decir, en vez de crear un enlace entre la raíz de la entidad A y la raíz de la entidad B, se deriva este enlace a partir del enlace entre el componente A-a_i, y el componente B-b_j.

Semántica de navegación por defecto

El uso real de un hipertexto está muy condicionado por la semántica de navegación, que indica como mostrar los objetos de la aplicación, sus enlaces, y los resultados obtenidos al activarlos. El modelo HDM se considera como un modelo estático, al que hay que superponer una semántica de navegación (compatible con él) para poder visionar el hipertexto. HDM define una *semántica de navegación por defecto*. Bajo esta semántica las unidades se presentan al usuario como los nodos del hipertexto, pudiendo haber solo un nodo activo cada vez (es decir, de forma similar a lo prescrito por el modelo HAM). Como consecuencia los lectores solo pueden percibir enlaces entre unidades, y por tanto, los enlaces navegables solo pueden establecerse entre unidades. Ya que en HDM solamente los enlaces de perspectiva conectan unidades, mientras que los estructurales y de aplicación se establecen entre componentes y/o entidades, estos últimos deben ser convenientemente trasladados a enlaces entre unidades. El modelo denomina *concretos* a los enlaces entre unidades, y *abstractos* al resto.

Para pasar de los enlaces abstractos a los concretos, HDM se apoya en el concepto de *representante por defecto* para cada componente o entidad. Para definir dicho representante es necesario introducir una *perspectiva por defecto* para cada tipo de entidad, y asumir que el representante para un componente es su unidad bajo la perspectiva por defecto, y que el representante para una entidad es el representante por defecto de su componente raíz. Por lo tanto el componente raíz de una entidad, en su perspectiva por defecto, representa a esa entidad. En el ejemplo de la Figura 2.25, y supuesto que la perspectiva por defecto es español, el representante por defecto es el ejercicio completo en español.

En base a este concepto, los enlaces de aplicación entre entidades se trasladan en enlaces concretos bajo sus representantes por defecto. De esta forma si la entidad ejercicio 8 enlaza con la entidad ejercicio 19, este enlace abstracto se concreta en un enlace entre sus representantes por defecto, es decir, entre ambos ejercicios.

Por otro lado cada enlace de aplicación entre componentes se traduce a un conjunto de enlaces concretos conectando cada unidad (bajo cualquier perspectiva) del componente origen con el representante por defecto del componente destino. Por ejemplo, si el componente ejercicio 8/enunciado conecta con el componente ejercicio 22/enunciado, concretamos este enlace en dos, uno de la unidad ejercicio 8/enunciado/español con la unidad ejercicio 22/enunciado/español, y el otro de la unidad ejercicio 8/enunciado/inglés con la unidad ejercicio 22/enunciado/español.

En cuanto a las *anclas* el modelo HDM las considera como simples botones asociados a los enlaces. Estas anclas pueden representar un enlace n-ario, posibilidad representada en el modelo mediante el concepto de *selector* (del inglés *chooser*), menú que permite seleccionar el destino concreto.

2.5.1.2 Ejemplo

El ejemplo que venimos manejando es el típico ejemplo que invalida el uso de modelos relacionales. Para empezar el modelo HDM no contempla la posibilidad de representar los contenidos dinámicamente generados. Si nos restringimos a los contenidos estáticos ¿cuál es la relación que existe entre estos?. Entre la foto del españa y las ciudades si que parece haber una relación de contenido/continente. Pero ¿y entre la biografía y españa?, ¿y entre la biografía y la docencia?, ¿y entre docencia y asignaturas? ¿Deberíamos tener una entidad Docencia que se relacionase con la entidad Biografía? ¿Y las biografías de los que no son profesores? ¿Debemos considerar una entidad para cada posible profesión de cada persona? ¿Consideramos una única entidad denominada Profesión? En ese caso ¿cómo se relacionaría la docencia con las asignaturas? Es decir, en este ejemplo no aparecen entidades lo suficientemente definidas como para caracterizar un tipo de entidad. La única opción, que es una aberración desde el punto de vista relacional, es considerar como entidades aquellas que van a aparecer en determinadas ventanas de determinadas pantallas. Esto también es una aberración desde el punto de vista de los sistemas de representación hipermedia, ya que estamos ligando inexorablemente el nivel de almacenamiento con el de presentación. Además, aunque elijamos esta opción, el modelo HDM no va a ser capaz de caracterizar el contexto del esquema navegacional. Por esto elegimos la opción "no aberrante" caracterizada por la Figura 2.26. Nótese que este esquema va a imposibilitar representar las múltiples relaciones que tiene biografía con docencia.

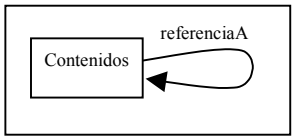

Figura 2.26 Esquema HDM para el ejemplo

Nótese que el único enlace abstracto que hay es referenciaA. Vamos a suponer que hay una única perspectiva por defecto, que es por ejemplo español. Con estas suposiciones la Figura 2.27 representa la instancia del esquema anterior.

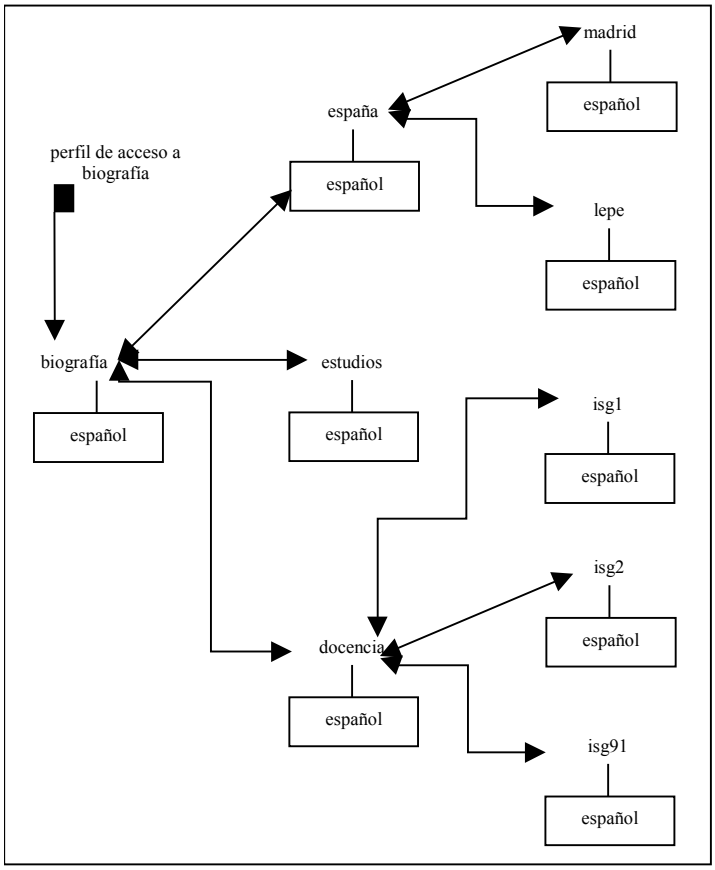

Figura 2.27 Instancia del esquema anterior

Esta instancia es extremadamente sencilla, ya que las entidades no están formadas por componentes, sino por una simple unidad (técnicamente se puede suponer que hay un componente entre la entidad y la unidad, pero la Figura 2.27 obvia dicho componente en aras de una mayor simplicidad). Por tanto no hay enlaces de perspectiva, ni estructurales. Solamente hay enlaces de aplicación entre entidades, que evidentemente, se concretan bajo la semántica de navegación por defecto en enlaces entre las unidades que componen las entidades. Nótese la equivalencia de la instancia del esquema con la representación de un hipertexto proporcionada por HAM (Figura 2.10). En cuanto a la semántica de navegación por defecto esta mostraría cada entidad en una pantalla, en un selector los enlaces con el resto de entidades.

2.5.1.3 Discusión y comparativa

El modelo HDM caracteriza los contenidos según el modelo E-R, y luego hace una transformación, más o menos compleja, al modelo HAM. Es por tanto un modelo muy limitado, ya que no solamente hay que hacer la traducción de un diseño E-R al modelo de grafos, sino que además solamente proporciona la semántica de navegación de HAM

Separación de contenidos, navegación y presentación. Este modelo se encarga casi exclusivamente de modelar la estructura de los componentes del nivel de composición interna, optando por una organización relacional de los mismos. El nivel de almacenamiento, aunque teóricamente se representa por enlaces de aplicación entre los elementos del nivel de composición interna (las entidades), en la práctica (es decir en la traducción de enlaces abstractos a enlaces concretos) es equivalente a un modelo de grafos. Nótese que en principio los enlaces de este nivel se definen entre entidades y esquemas, teniendo que concretarse luego en enlaces entre contenidos reales. El cómo se concreta es lo que el modelo llama semántica de navegación por defecto. Bajo esta semántica las entidades pierden su estructura al pasar al nivel de presentación (se traducen por un nodo que contiene todas sus unidades). Los enlaces estructurales, por tanto, en la funcionan como amalgamadores de contenidos de la entidad. Los enlaces de aplicación que partían de los componentes y de las propias entidades se traducen por botones asociados a la pantalla que presenta la entidad, y que permiten navegar a otras entidades. En este escenario los enlaces de perspectiva, se traducen por botones que permiten cambiar entre distintas visiones de la misma entidad. El modelo no proporciona nivel de presentación. Solo incluye una descripción de una semántica de navegación por defecto, equivalente a la proporcionada por HAM.

Contexto. Este modelo es muy limitado y no considera la idea de contexto, ni a nivel esquema/instancia ni a nivel semántica de navegación por defecto. Evidentemente se podría proporcionar una semántica de navegación que permitiese representar la noción de contexto, pero esta semántica también debería indicar como concretar los enlaces abstractos que aparecen en el modelo relacional.

Sincronización. El modelo no proporciona ningún soporte a la sincronización. El comentario sobre posibles ampliaciones es el mismo que el del contexto.

Formalización de la semántica de navegación. El modelo no proporciona ningún formalismo para representar el esquema navegacional (salvo el inducido por las relaciones existentes entre las entidades). Lo más que fija es la semántica de navegación mínima, de la cual, se puede deducir un esquema navegacional similar al proporcionado por el modelo HAM. Incluso esta deducción no es inmediata, ya que hay que eliminar la estructura impuesta a las entidades, y concretar los enlaces abstractos. Por otro lado la semántica de navegación viene determinada por la semántica de navegación por defecto, equivalente como hemos visto a la proporcionada por el modelo HAM. Esto posibilitaría la construcción de entornos de generación automática de prototipos/aplicaciones, aspecto no cubierto por los autores del modelo.

Organización relacional de los contenidos. Este modelo incluye una organización relacional de los contenidos, pero a diferencia del modelo RMDM, esta organización no se corresponde de forma clara con un esquema navegacional. Esto se debe a que en la práctica la complejidad interna de las entidades, y de los enlaces entre estas, no tiene una traducción directa con una estructura navegacional. El modelo RMDM supondrá entidades planas (sin organización interna), y usará las relaciones entre estas para determinar el esquema navegacional de la aplicación.

Gestión explícita de eventos. El modelo no soporta ningún tipo de gestión de eventos. Realmente ni siquiera soporta un modelo claro de semántica de navegación, solamente una heurística a seguir por aquellos implementadores que utilicen este modelo para diseñar aplicaciones hipermedia.

Generación dinámica de contenidos. El modelo no trata de caracterizar los contenidos generados dinámicamente.

Gestión de usuarios y seguridad. Resulta curioso, que aunque se trata de un modelo formulado en el contexto de bases de datos, este no considera ninguna gestión explícita de usuarios ni de seguridad.

Distinción de enlaces a nivel contenidos y esquema navegacional. El modelo realmente presenta cierta inconsistencia entre el concepto de enlace entre contenidos, y su traducción a semántica de navegación. Los enlaces de perspectiva simplemente indican que hay varias vistas de los contenidos. Los enlaces estructurales sirven para especificar la estructura de los contenidos de las entidades. Los enlaces de la aplicación son los enlaces reales entre entidades. Estos se establecen entre componentes y entidades, es decir, entre la parte de la entidad que no tiene contenidos (las unidades). Sin embargo, a la hora de fijar una semántica de navegación por defecto, los únicos enlaces válidos son los establecidos entre contenidos (unidades). Por tanto hay que hacer una asimilación de los enlaces entre estructuras a enlaces entre contenidos reales. Esto no supone una doble concepción de enlaces. Simplemente pone de manifiesto que el tener definidos los enlaces al nivel esquema de los contenidos presenta la dificultad de tener que traducirlos a contenidos concretos, pero en principio, tanto los enlaces de aplicación concretos como abstractos representan el mismo enlace navegacional.

Técnicas orientadas a objetos. El modelo no se encarga de representar ninguna información ajena a la estructura de los contenidos, y sus relaciones entre ellos. Por esto no incluye técnicas de diseño orientado a objetos.

2.5.2 El modelo de datos de gestión de relaciones (RMDM)

2.5.2.1 Presentación

Introducción

El modelo de datos para la gestión de relaciones, RMDM (del inglés *Relationship Managemente Data Model)* [Isakowitz 95], es un modelo hipermedia de corte relacional para describir aplicaciones hipermedia. Forma parte de la metodología de gestión de relaciones, RMM (del inglés *Relationship Management Methodology*) [Isakowitz 95], un modelo de proceso para la construcción de aplicaciones hipermedia, que se apoya en el modelo de datos anterior. En esta parte solamente nos centraremos en el modelo de datos, encargado de representar las aplicaciones hipermedia. En el Capítulo 3, veremos en profundidad cual es el modelo de proceso que engloba a este modelo de datos.

El tipo de aplicaciones para las que RMM es más adecuada, muestra una estructura regular en el dominio de interés. De esta forma hay clases de objetos, relaciones entre clases, y múltiples instancias de objetos dentro de cada clase. Por lo tanto, el modelo de datos se puede aplicar bastante bien a catálogos de productos e interfaces de sistemas de gestión de bases de datos. También puede ser aplicado kioscos de información. Pero en la medida que la información no sea susceptible de admitir una representación relacional, como la mayoría de páginas HTML, o aquellas aplicaciones hipermedia cuyos contenidos tengan una estructura distinta de la relacional (como un libro electrónico), el modelo no proporciona una representación conveniente de los datos [Isakowitz 95]. Esto no es una deficiencia del modelo de datos. Simplemente es un modelo muy específico que se adecua bien a un amplio pero determinado espectro de aplicaciones hipermedia.

Bajo este prisma de vista, los contenidos de las aplicaciones hipermedia van a poseer una estructura relacional. Las entidades (en este caso sin estructura) serán los contenidos de la aplicación, mientras que las relaciones entre entidades serán utilizadas para inducir el esquema navegacional de la aplicación.

El modelo de datos

RMDM proporciona un lenguaje para describir las aplicaciones hipermedia, entendidas como objetos de información y mecanismos de navegación entre estos. La Figura 2.28 muestra las primitivas de modelado de

RMDM. Las primitivas del dominio, modelan la información del dominio de aplicación. Los tipos de entidades y sus atributos representan objetos abstractos o físicos, tales como personas o cuentas de bancos. Las relaciones de asociación, que pueden ser una-a-una, o una-a-muchas, describen asociaciones entre distintos tipos de entidades. Al igual que en el modelado, una relación muchas-a-muchas se factoriza en dos relaciones una-a-muchas [Elmasri 97].
Ya que las entidades pueden estar formadas por un gran número de atributos de distinta naturaleza (por ejemplo, salario, fecha de nacimiento, fotografía) en la práctica puede ser necesario mostrar cierto número de estos atributos, en lugar de todos a la vez. Con este fin, los atributos se agrupan en *porciones*. Por ejemplo, una entidad persona con atributos nombre, edad, foto y biografía puede tener una porción general que contenga el nombre, edad y fotografía, y una porción biografía con la biografía. De esta forma, cada instancia de la entidad persona estará representada por dos porciones, y si la aplicación lo soporta, un usuario puede elegir entre una vista u otra. Nótese que el concepto de porción está relacionado con el de perspectiva del modelo HDM, ya que ambos proporcionan distintas visiones de una misma entidad.

En RMDM la navegación está representada por seis primitivas de acceso. Los *enlaces unidireccional* y *bidireccional*, se utilizan para especificar accesos entre porciones de una entidad (algo similar a los enlaces estructurales de

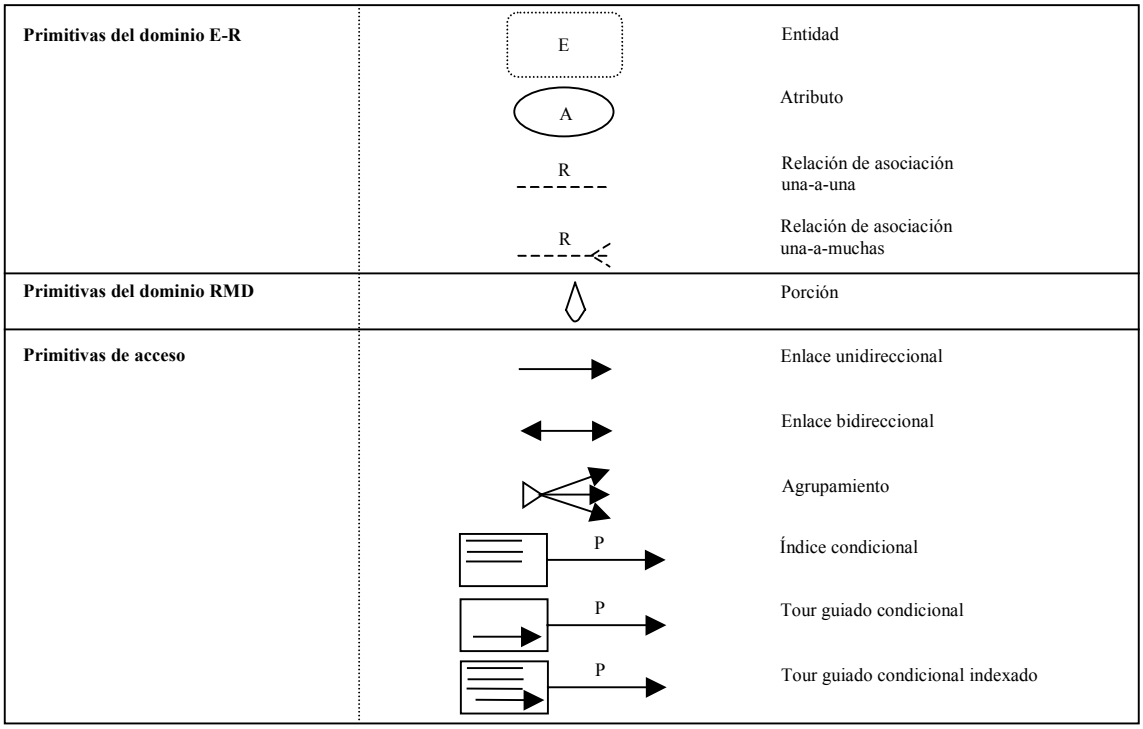

Figura 2.28. Primitivas de modelado RMDM

HDM), pudiéndose utilizar solamente en las fronteras de una entidad. La navegación entre distintas entidades viene representada mediante *índices*, *tours guiados*, y *agrupamientos*. Estas últimas primitivas se instancian sobre relaciones existentes entre entidades, siendo por tanto una forma de acceder a las entidades en base a sus relaciones con las otras. Evidentemente alguna entidad deberá ser la inicial. Con este fin se utilizan las tres primeras primitivas. Un *índice*, actúa como una tabla de contenidos a una lista de instancias de entidad, proporcionando acceso directo a cada una de ellas. Un *tour guiado* implementa un camino lineal entre una colección de elementos, permitiendo al usuario moverse hacía delante o hacía atrás en este camino. Existen variaciones de los tours guiados. Por ejemplo, un tour guiado *circular* enlaza el último elemento con el primero; un tour guiado con *vuelta al principal* tiene un nodo distinguido que contiene información sobre el propio tour guiado, que actúa simultáneamente como el principio y el final del tour; y un tour guiado con *entrada y salida* tiene distintos nodos para la entrada y la salida.

La construcción de *agrupamiento* es un mecanismo de menús que permite el acceso a otras partes del documento hipermedia. Un ejemplo típico de agrupamiento es la pantalla inicial de muchas aplicaciones, que sirve como plataforma de acceso a otras características tales como índices y tours guiados. Los *índices* son tipos especiales de agrupamientos que soportan condiciones. Las condiciones o predicados lógicos que etiquetan a los índices y tours guiados determinan que instancias de una entidad son accesibles desde esa construcción. La Figura 2.29 ilustra estos conceptos.

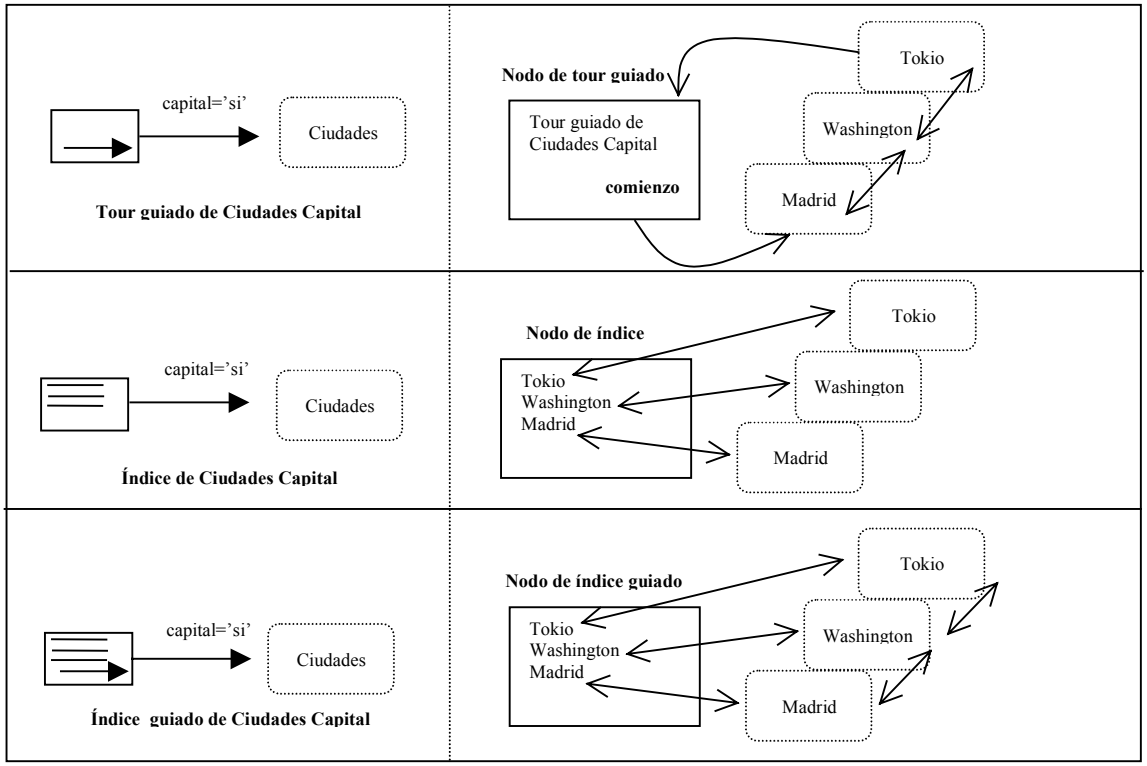

Figura 2.29 Primitivas de acceso RMDM

2.5.2.2 Ejemplo

Veamos como representar nuestro ejemplo según el modelo RMDM. Al igual que en el modelo HDM, la naturaleza de los contenidos no se ajusta demasiado bien al modelo relacional. Tampoco es capaz de representar contenidos generados dinámicamente. Además ahora aparece un problema añadido. En el modelo HDM los enlaces concretos se deducían automáticamente de las relaciones entre entidades, pero ahora es necesario explicitar las relaciones entre las entidades que deseamos aparezcan como hiperenlaces. Por lo tanto necesitamos incluir una relación por cada "nivel de relación" que exista en el grafo de enlaces de contenidos. Como ya comentamos en el caso de HDM esto es una aberración desde el punto de vista relacional, ya que estamos estructurando el dominio en base a las relaciones navegacionales que existen entre los individuos, y no en base a las características comunes de estos que provocarían su unión en un tipo de entidad. También es aberrante desde el punto de vista hipertextual al ligar el nivel de almacenamiento con el de presentación. La Figura 2.30 ilustra esta situación.

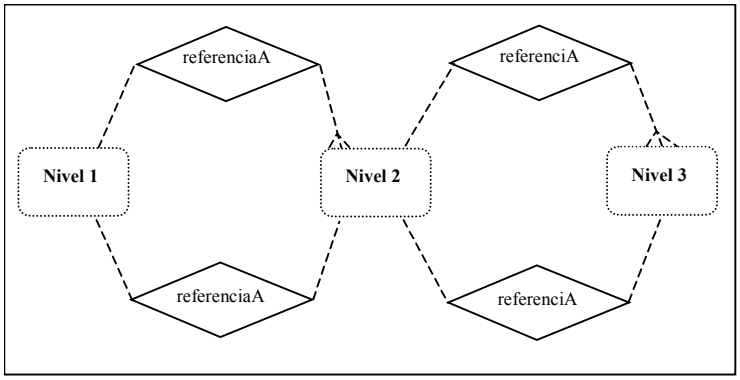

Figura 2.30 Diagrama (aberrante) E-R del ejemplo

En las entidades de Nivel 1 se encuentra la biografía. Las entidades de Nivel 2 son la foto de españa, los estudios, y la docencia. Finalmente las entidades de Nivel 3 son madrid, lepe, isg1, isg2, e isg91. Nótese como hemos perdido los múltiples enlaces que existen entre la biografía y la docencia, y que las primitivas de acceso RMDM no son suficientes para representar la semántica navegacional requerida. En efecto, si utilizamos las relaciones entre entidades como vehículo para insertar las primitivas de acceso tenemos la Figura 2.31, figura en la que se proporciona una pantalla a cada entidad, sin ningún tipo de contexto.

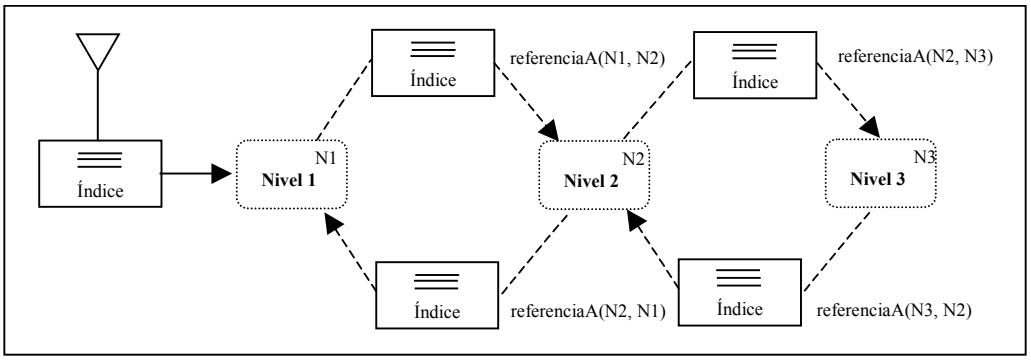

Figura 2.31 Diagrama RMDM del ejemplo

Según esta, y según la interpretación de los índices (Figura 2.29) el diagrama anterior se traduce a una estructura de nodos como la representada por el modelo HAM (Figura 2.10). Es decir, en una pantalla aparecería la biografía con tres botones para acceder a la foto de españa, los estudios o la docencia. Si se seleccionase la foto de españa, por ejemplo, esta aparecería en otra pantalla con dos botones, uno que permite acceder a madrid, y otro a lepe. La condición referenciaA(N2, N3), garantiza que en la pantalla asociada a las fotos solo aparecen los botones que enlazan con las ciudades. Nótese que al igual que en HDM hemos perdido la posibilidad de que la biografía enlace de cuatro formas distintas con la docencia. Esto se debe a que las relaciones se establecen entre entidades, y no entre entidades y sus anclas, siendo estas últimas meros botones asociados a la pantalla donde aparece la entidad.

2.5.2.3 Discusión y comparativa

El modelo RMDM es un modelo conceptualmente muy claro. Organiza el dominio de la aplicación según un modelo E-R, y a continuación utiliza las relaciones definidas en este modelo para generar una estructura de nodos equivalente a la propuesta por HAM. Esta aproximación solo presenta dos inconvenientes: no es aplicable a dominios no estructurables según el modelo E-R , salvo organizaciones "aberrantes" (como la del ejemplo), y no se puede establecer un enlace entre entidades que no tengan establecidas una relación formal. Además la traducción del esquema relacional, al esquema navegacional solo permite representaciones similares a las de la máquina abstracta de hipertexto.

Separación de contenidos, navegación y presentación. En este modelo en nivel de composición interna viene determinado por un esquema E-R. Ahora si que se supone una cierta estructuración de los datos (los atributos de cada entidad), y estos mantienen relaciones, en principio independientes con el esquema navegacional asociado. En la práctica esto no es así, ya que el nivel de almacenamiento se deduce casi por completo del nivel de composición interna. Es decir, los enlaces entre nodos de este nivel se deducen de las relaciones existentes entres las entidades del nivel anterior. Por lo demás, el nivel de almacenamiento es equivalente a un grafo dirigido. Las anclas son meros botones asociados a cada nodo en función de los enlaces que partan de él. El nivel de presentación se deriva totalmente de la estructura de grafo proporcionada por el nivel de almacenamiento, y el modelo no especifica ninguna primitiva de especificación de presentación.

Contexto. El modelo no permite representar la noción de contexto, ya que como hemos comentado, el nivel de almacenamiento es equivalente a un grafo. Probablemente no sería complejo extender las primitivas de acceso para representar el concepto de frame.

Sincronización. El modelo no proporciona ningún soporte a la sincronización. Probablemente no sería muy complejo una extensión de las primitivas de acceso para permitir sincronización.

Formalización semántica navegacional. En este caso el modelo proporciona una representación relacional del esquema navegacional, fácilmente traducible a un grafo dirigido. Por lo tanto, el esquema navegacional se encuentra implícito en la estructura real. Nótese que este formato relacional presenta la gran ventaja de no tener que mostrar en un diagrama todos los enlaces existentes entre los contenidos reales. Simplemente se indica que hay una relación, y las entidades unidas por esa relación serán las conectadas por el enlace. Esto presenta el inconveniente de que solo se pueden establecer enlaces entre entidades si existe una relación en el dominio. Por otro lado la semántica de navegación se determina en función de las estructuras de acceso. En cualquier caso si que permitiría la generación automática de prototipos/aplicaciones en base a los diagramas RMDM, eso sí, sin incluir información sobre sincronización ni contexto.

Organización relacional de los contenidos. El modelo RMDM opta por una organización relacional tradicional de los contenidos. Frente al modelo HDM, las entidades si son relacionales puras, es decir, no tienen ninguna estructura interna (salvo la proporcionada por los atributos). Al igual que antes las relaciones del dominio se traducen a enlaces, pero ahora no surge el inconveniente de concretar los enlaces abstractos a enlaces concretos entre contenidos. Las entidades son planas, y toda la entidad conecta con la totalidad de otra entidad. Tampoco presenta el problema concretar los enlaces estructurales, ya que ahora las entidades no poseen. Finalmente, los enlaces de perspectiva del modelo HDM están representados en RMDM por las porciones, que permiten seleccionar determinados atributos de las entidades. El inconveniente de la organización relacional de los contenidos, es que existen dominios, en los cuales una organización relacional no es la más adecuada. Dentro de este dominio podemos enclavar todos los documentos organizables mediante lenguajes de marcas, organización alternativa, y en principio independiente de la relacional. Esto no quiere decir que no existan mecanismos, más o menos automáticos, para convertir de un formato a otro. Solamente quiere decir, que hay dominios en los cuales una organización documental es más sencilla y/o directa que una organización relacional (por ejemplo un libro).

Gestión explícita de eventos. El modelo simplemente se preocupa de modelar el dominio y traducirlo a un grafo dirigido. En este contexto de aplicación los eventos ni siquiera se consideran.

Generación dinámica de contenidos. El modelo concibe una transición dinámica del diagrama RMDM que describe a la aplicación hipermedia a sus instancias concretas. Es decir, en el caso del ejemplo los enlaces de españa con madrid y lepe se generarían en tiempo de ejecución como resultado de una pregunta (referenciaA(españa, N3)), a la base de datos. El problema es aplicar este esquema a contenidos que no se ajustan al esquema relacional, como puede ser un formulario de entrada de datos. En cualquier caso, el esquema dinámico solo se presenta a nivel teórico, debiéndose resolver su implementación en la práctica.

Gestión de usuarios y seguridad. Al igual que el modelo HDM resulta muy curioso que un modelo con tan fuertes fundamentos en el mundo de las bases de datos no incluya ningún soporte para la gestión de usuarios ni la seguridad.

Distinción de enlaces a nivel contenidos y esquema navegacional. El modelo presenta una cierta distinción entre los enlaces a nivel de contenidos (relaciones entre entidades) y los enlaces del esquema navegacional (agrupamientos, índices, tours guiados y tours guiados indexados), lo que sucede es que en la práctica es esquema navegacional se deduce casi en su totalidad del esquema relacional de los contenidos. La ventaja de esta aproximación, es la ocultación de todos los enlaces a nivel de contenidos (entre entidades) mediante las primitivas de accesos que se establecen entre tipos de entidades. El inconveniente radica que en la práctica, estos enlaces a nivel de entidades completas se traducen en referencias de la entidad destino a su entidad origen.

Técnicas orientadas a objetos. El modelo no considera que en la aplicación pueda haber nada que no se derive totalmente del diagrama E-R de los contenidos, y por tanto no incluye mecanismos de representación genéricos.

2.6 Sistemas de representación con tratamiento explícito de eventos

Terminamos el estudio de los sistemas de representación hipermedia con un par formalismos que consideran necesario modelar características de la aplicación hipermedia más allá de la estructura navegacional asignada a está. Con este motivo el modelo hipermedia Labyrinth puede incorporar un lenguaje de programación de propósito general que permite representar cualquier actividad computacional más allá de la activación de un enlace. Con el mismo fin la metodología OOHDM incluye toda la potencia de las técnicas de análisis y diseño orientadas a objetos para ser capaz de representar cualquier evento (más allá de la activación de un enlace) que aparezcan en una aplicación hipermedia. Nótese que ambas aproximaciones son en principio distintas, ya que Labyrinth (modelo hipermedia) opta por proporcionar el código de estas actividades computacionales, mientras que OOHDM (metodología de diseño) opta por proporcionar una representación orientada a objetos de dichas actividades.

2.6.1 El modelo Labyrinth

2.6.1.1 Presentación

Introducción

El modelo Labyrinth [Díaz 94], [Díaz 01], es el único junto al modelo OOHDM capaz de responder a eventos genéricos, o en términos propios del modelo, capaz de especificar *comportamientos dinámicos*. El modelo Labyrinth es un modelo para el diseño de aplicaciones hipermedia, siendo sus características más relevantes:

- Permitir diseñar aplicaciones hipermedia independientes de la plataforma.
- Permitir la categorización, generalización y abstracción de información dispersa y heterogénea en distintos niveles interconectados.
- Permite la representación de todo tipo de información multimedia, temporización, sincronización, etc.
- Permite la creación de vistas personales en hiperdocumentos multiusuarios para grupos y usuarios individuales.
- Permite la inclusión de mecanismos de seguridad según la definición de seguridad dada por el DTI (*Department of Trade and Industry of the USA*).

El modelo se divide en dos partes, una *estática*, y otra *dinámica*, caracterizadas por una parte declarativa del modelo y otra basada en especificaciones de eventos. El modelo declarativo puede utilizarse para representar aquellas características de los hiperdocumentos que se mantienen estables, en el sentido de que no dependen de ningún factor externo, y tienen una existencia significativa por sí mismas. Sin embargo hoy en día las aplicaciones hipermedia incluyen estructuras, contenidos, o interfaces que se generan en tiempo de ejecución. En estos casos, el modelo de eventos debe ser ampliado con especificaciones basadas en eventos, donde se establezcan las condiciones que restrinjan la creación y objetos y el proceso de creación de los mismos.

Además, al igual que OOHDM o RMDM, el modelo va asociado con una metodología de desarrollo de aplicaciones hipermedia denominada Ariadne. Estudiaremos está metodología en detalle en el Capítulo 3.

Labyrinth

El modelo Labyrinth representa a una aplicación hipermedia a través de un *Hiperdocumento Básico*, donde se especifican cierto número de elementos para definir la estructura y el comportamiento de una aplicación. Además cada usuario o grupo de usuarios puede tener un *Hiperdocumento Personalizado*, donde los usuarios pueden adaptar componentes del Hiperdocumento Básico para sus propios requisitos, o crear uno nuevo.

Por tanto, un *Hiperdocumento* (HD) se define como la unión de un Hiperdocumento Básico (HD^B) y una serie de Hiperdocumentos Personalizados (HD^P), cada uno de los cuales pertenece a un grupo de usuarios. Es decir,

$$HD = HD^B \cup HD^P$$

donde,

$HD^B = (U, N, C, A, L, B, E, lo, al, el, ac)$

donde,

- U, es el conjunto de *usuarios* del hiperdocumento
- N, es el conjunto de *nodos* del hiperdocumento
- C, es el conjunto de *contenidos* del hiperdocumento
- A, es el conjunto de *anclas* del hiperdocumento
- L, es el conjunto de *enlaces* del hiperdocumento
- B, es el conjunto de *atributos* del hiperdocumento
- E, es el conjunto de *eventos* del hiperdocumento
- lo, es una función que determina la *localización* de un contenido en un nodo, es decir,
  $lo: \forall C_i \in C, \forall N_j \in N \mid i = 0,..., n, n \in \mathbb{N}, j = 0,..., m, m \in \mathbb{N}, lo(C_i, N_j) = \{ Posición_i, Tiempo_i \}$
  donde,
  $Posición_i$ es la posición del contenido en el nodo
  $Tiempo_i = \{Comienzo_i, Duración_i\}$ indica el momento el que el contenido se coloca en el nodo, y el intervalo de permanencia en él.

- al, es una función que asigna valores a la lista de *atributos* de un elemento, es decir,

  $al: \forall x \in (U \cup N \cup C \cup L), al(x) = \{NombreAtributo_i, Valor_i\}, \quad i = 0,..., n, n \in \mathbb{N}, NombreAtributo_i \in B_i$

- el, es una función asigna *eventos* a un elemento, es decir,

  $el: \forall x \in (N \cup C \cup L), el(x) = \{IdEvento_i, Prioridad_i\}, i = 0,..., n, \ n \in \mathbb{N}$

- ac, es una función que asigna la categoría de acceso de un elemento, a un usuario, es decir,

  $ac: \forall U_i \in U, \forall x \in (HD \cup N \cup C), ac(U_i, x) = CategoríaAcceso_i$

Además, cada usuario o grupo de usuarios puede tener un Hiperdocumento Personalizado en el cual los usuarios pueden crear un nuevo hiperdocumento, o modificar algunos componentes del Hiperdocumento Básico, adaptándolo a sus necesidades. Un Hiperdocumento Personalizado (HDP) se define:

$HD^P = \{ HD^P_i \mid i = 0, ..., n \in N \}$ donde
$HD^P_i = (IdUsuario, N^p, C^p, A^p, L^p, B^p, E^p, lo^p, al^p, el^p)$

donde IdUsuario es el identificador del usuario indicando el poseedor de la personalización (grupo o individual) y el resto de elementos son personalizaciones de algunos elementos definidos en el Hiperdocumento Básico. N^p, C^p, A^p, L^p, B^p, E^p, lo^p, al^p, el^p están compuestos por nuevos elementos y funciones o modificaciones de los mismos.

El modelo también incluye operaciones relacionadas con el manejo del hiperdocumento. Los conjuntos de usuarios, nodos, contenidos, anclas, enlaces, atributos y eventos pueden ser accedidos y actualizados en base a las necesidades del usuario. Por ejemplo, los usuarios pueden definir versiones de nodos y contenidos. Una nueva versión incluye todos los elementos asociados al nodo o a los contenidos (es decir, contenidos, anclas, enlaces, atributos y eventos), y es conectado automáticamente a las versiones previas y siguiente a través de un tipo especial de enlace (denominado *enlace de versión*) que puede ser modificado. Las operaciones relacionadas a las funciones del modelo, permiten asignar y modificar valores a sus elementos y obtener información acerca de los valores de las funciones. Cada operación incluye un control de seguridad para determinar si el usuario puede lleva a cabo la operación. La descripción detallada y formal de estas operaciones de manejo del hiperdocumento puede encontrarse en [Díaz 01]. A continuación profundizaremos en la descripción de un hiperdocumento según el modelo Labyrinth.

USUARIOS: $U_i = (IdUsuario_i, TipoUsuario_i, ListaUsuario_i)$

Los usuarios y grupos son tratados como un *usuario* e identificados por un IdUsuario alfanumérico, y por tanto, todo lo dicho para usuarios puede aplicarse a grupos de usuarios. Una etiqueta (TipoUsuario) especifica si el usuario es una persona o un grupo. Cada usuario tiene asociada una lista (ListaUsuario). Si el usuario es individual, la lista incluye los grupos en los que participa. Si el usuario es un grupo, la lista incluye el nombre de los miembros del grupo. Cuando un usuario con los privilegios correspondientes crea un hiperdocumento personalizado, dicho documento se identifica por el IdUsuario del usuario. La Figura 2.32 muestra las sentencias donde se definen dos grupos de usuarios, y un usuario individual.

```
{ESPAÑOL, Grupo, [Carmen, Dolores]}
{Estudiantes, Grupo, [Carmen, Mario]}
{Carmen, Individual, [ESPAÑOL, Estudiantes]}
```

Figura 2.32 Definición de usuarios

NODOS: N_i = ($IdNodo_i$, $CategoríaNodo_i$)

El *nodo* es un contenedor de información definido por un identificador (IdNodo) y caracterizado por las operaciones permitidas a los usuarios (CategoríaNodo). El contenido del nodo se define de manera independiente, y no se establecen restricciones sobre la naturaleza de los mismos. Los *nodos compuestos* son estructuras lógicas formadas por otros nodos y nodos compuestos. Estos nodos compuestos recogen los mecanismos de abstracción definidos en [Garg 88]: agregación y generalización. Los nodos compuestos también se definen como nodos, y los mecanismos de abstracción se definen a través de los enlaces. La Figura 2.33 recoge ejemplos de nodo.

```
{Don Quijote - Dulcinea, personalización}
{Concepto de Quijotismo, personalización}
{Cualidades morales de Don Quijote, personalización}
```

Figura 2.33 Ejemplo de nodos

En esta figura aparecen tres nodos que contendrán información sobre el Quijote. Se definen como personalización porque serán modificados por los usuarios.

CONTENIDOS: C_i = ($IdContenido_i$, $CategoríaContenido_i$, $TipoContenido_i$)

Un *contenido* es una pieza de información definida por un identificador (IdContenido) y caracterizada por las operaciones permitidas a los usuarios (CategoríaContenido). Los parámetros Tipo y Unidades de TipoContenido se utilizan para describir de que tipo de información es el contenido. Se permite todo tipo de información (texto, gráficos, vídeo, programas, sonido, etc.), pero se deben especificar las unidades de dimensión (pixels, caracteres, frames, etc.). Un contenido puede asignarse en cualquier posición de un nodo a través de la función de localización lo. La Figura 2.34 muestra ejemplos de contenidos.

```
{Discusión sobre Don Quijote – Dulcinea, presentación, [texto, [palabra]]}
{Ballet, presentación, [vídeo, [frame]]}
{Comentario 1990/09/19, edición, [texto, [carácter]]}
```

Figura 2.34 Ejemplo de contenidos

En este ejemplo tenemos tres contenidos. El contenido `Discusión sobre Don Quijote – Dulcinea` es un análisis de la relación entre los personajes de la novela. Se define como un texto medido en palabras, y solo puede ser presentado. `Ballet` es un vídeo cuya longitud se mide en `frames`. El último contenido son los comentarios personales de algún usuario. Es un texto editable cuya longitud se mide en caracteres.

ANCLAS: A_i = ($IdAncla_i$, $IdNodo_i$, $IdContenido_i$, $PosAncla_i$)

Un *ancla* se define como un identificador alfanumérico (IdAncla) y determina un lugar de referencia dentro de un nodo y/o contenido. Estos últimos se identifican utilizando los parámetros IdNodo e IdContenido. Posición y Extensión en PosAncla, especifican la localización y extensión del ancla. La Figura 2.35 muestra ejemplos de anclas.

```
{Pi1, Cualidades morales de Don Quijote, -1, }
{Pd1, Concepto de Quijotismo, -1, }
{Pd2, Don Quijote – Dulcinea, -1, }
{Ancla Ballet 1, -1, Comentario 1990/09/19, ["Strauss", 1] }
{Ancla Ballet 2, -1, Ballet, [N, D] }
```

Figura 2.35 Ejemplo de anclas

`Pi1` es un ancla definida sobre el nodo `Cualidades morales de Don Quijote`. Será utilizada como punto de inicio de un enlace entre este nodo y los nodos `Concepto de Quijotismo` y `Don Quijote – Dulcinea` (puntos de destino `Pd1` y `Pd2` respectivamente). `Ancla Ballet 1` es un ancla definida sobre la palabra `Strauss` del contenido `Comentario 1990/09/19`, cuya extensión es una palabra. El ancla se utilizará como punto de inicio de un enlace del frame `N` del vídeo `Ballet` (punto destino `Ancla Ballet 2`), cuya extensión será `D` frames de vídeo.

ENLACES: L_i = (IdEnlace$_i$, OrigenEnlace$_i$, DestinoEnlace$_i$)

Un *enlace* es una conexión etiquetada uni o bidireccional entre dos conjuntos de puntos, orígenes y destino en la aplicación hipermedia. Las anclas se utilizan para etiquetar el origen y el destino del enlace a través de los atributos OrigenEnlace y DestinoEnlace. El modelo incluye cuatro tipos de enlaces: *Referenciales*, de *Agregación*, de *Generalización*, y de *Versión*. Los enlaces Referenciales sirven para representar conexiones arbitrarias entre elementos (nodos o contenidos), es decir, los enlaces hipermedia. Los enlaces de Agregación permiten establecer nodos compuestos. Los enlaces de Generalización sirven para definir características comunes a familias de nodos. Finalmente los enlaces de Versión permiten incluir enlaces a las versiones sucesivas de un elemento. La Figura 2.36 muestra ejemplos de enlaces.

```
{Enlace 1, [Pi1], [Pd1, Pd2]}
{Enlace 2, [Ancla Ballet 1], [Ancla Ballet 2]}
```

Figura 2.36 Ejemplo de enlaces

En esta figura podemos ver el `Enlace 1` como un enlace n-ario conectando el nodo `Cualidades morales de Don Quijote` con los nodos `Concepto de Quijotismo` y `Don Quijote – Dulcinea` mediante las anclas origen `Pi1` y las anclas destino `Pd1` y `Pd2`. `Enlace 2` es un enlace de una palabra de un texto a un frame de un vídeo.

ATRIBUTOS: B_i = (NombreAtributo$_i$, Valor$_i$)

Los *atributos* son propiedades que se pueden asociar a usuarios, nodos, contenidos y enlaces mediante la función de valor de atributos al. Un atributo se representa mediante un Valor asignado por defecto durante la fase de definición y modificable a través de la función al. Es decir, estas definiciones funcionan como valores por defecto para los atributos. No hay restricciones en el número de atributos para cada elemento del modelo, aunque cada usuario, nodo, contenido y enlace debe tener al menos los siguientes atributos: Usuario(Etiqueta), Nodo(Etiqueta, Autor), Contenido(Etiqueta, Autor), Enlace(Etiqueta, Autor, TipoEnlace, Dirección). Etiqueta es un nombre o descripción que contiene información semántica del elemento. Autor indica el usuario que creó el elemento. TipoEnlace indica si el enlace es Referencial, Versión, Generalización o Agregación. Dirección diferencia entre enlaces uni y bidireccionales. La Figura 2.37 muestra ejemplos de atributos.

```
{TipoEnlace, "Referencial"}
{Dirección, "Bidireccional"}
```

Figura 2.37 Ejemplo de atributos

Este ejemplo fija los valores por defecto de los atributos TipoEnlace y Dirección a Referencial y Bidireccional, respectivamente.

EVENTOS: E_i = (IdEvento$_i$, Condición$_i$, ListaAcción$_i$)

Un *evento* se define como una condición cuyo cumplimiento conlleva una reacción bien definida del sistema. La condición, especificada en el parámetro Condición, es una expresión booleana de estímulos externos (por ejemplo la presión del ratón), estímulos internos (por ejemplo el final de una película), variables independientes (por ejemplo el tiempo), y/o valores constantes. Las reacciones activadas por el evento y recogidas en la lista ListaAcción son operaciones que deben ser llevadas a cabo por los elementos del hiperdocumento. Los eventos se ligan a nodos, contenidos o enlaces a través de la función de listado de eventos el. La Figura 2.38 recoge un ejemplo de eventos.

```
{Evento A, "nodo activado" AND "usuario pertenece al grupo de
estudiantes", ["reproducir la película del Ballet"]}
```

Figura 2.38 Ejemplo de evento

En este ejemplo cuando un usuario que pertenece al grupo de estudiantes llega a un nodo específico, la película se reproduce la película Ballet. Este evento no depende de ningún nodo particular, pero puede ser asociado a cualquiera, de tal forma que será evaluado cuando el nodo seleccionado sea accedido.

FUNCIÓN DE LOCALIZACIÓN: lo(C_i, N_i) = {Posición$_i$, Tiempo$_i$}

La *función de localización* asigna contenidos a un nodo. Dentro del nodo, el contenido se localiza en Posición, comienza en Comienzo y continúa durante Tiempo unidades. Esta función también permite especificar características de alineación espacial y sincronización temporal de los contenidos. Las *alineaciones* se especifican mediante restricciones lógicas que etiquetan arcos entre contenidos. De esta forma podemos especificar que dos contenidos son disjuntos, que A esta a la izquierda de B, y que la distancia en el eje vertical es cero mediante la expresión (disjuntos, izquierda, (-, 0)) etiquetando el arco que conecta A y B. Si queremos expresar sincronización entre contenidos Labyrinth permite utilizar arcos de sincronización similares a los utilizados por el modelo Amsterdam. La Figura 2.39 muestra un ejemplo de localización.

```
lo(Discusión sobre Don Quijote – Dulcinea, Don Quijote – Dulcinea) = (a, –1)
```

Figura 2.39 Ejemplo de localización

En el ejemplo se localiza el contenido `Discusión sobre Don Quijote – Dulcinea` en la posición a del nodo `Don Quijote – Dulcinea`. El `–1` indica que el texto no tiene limitaciones temporales y que será presentado desde que se llega al nodo y hasta que se abandone.

FUNCIÓN DE VALOR DE ATRIBUTO: al(x) = {NombreAtributo$_i$, Valor$_i$}

La *función de valor de atributo* asigna un valor a un atributo, ligándolo al usuario, nodo, contenido o enlace. El valor por defecto se especifica en la definición de atributos, y puede ser cambiado dándole un nuevo Valor. Como todos los elementos tienen al menos un atributo obligatorio, esta función entra en juego. Si no se asignan explícitamente valores de atributos cuando se crea un nuevo nodo, contenido, usuario o enlaces se toman los valores por defecto. La Figura 2.40 muestra ejemplos.

```
al(Enlace 1) = { (Etiqueta, "Partes de la discusión"), (Autor, "Carmen"),
                 (TipoEnlace, "Agregación"), (Dirección, Valor por defecto) }
al(Enlace 2) = { (Etiqueta, "Ballet Strauss"), (Autor, "Carmen"),
                 (TipoEnlace, Valor por defecto), (Dirección, Valor por defecto) }
```

Figura 2.40 Ejemplos de asignación de valor de atributo

En este ejemplo se asignan valores a los atributos de dos enlaces.

FUNCIÓN DE VALOR DE EVENTO: el(x) = {IdEvento$_i$, Prioridad$_i$}

La *función de valor de evento* asigna una prioridad a un evento y lo asocia a un nodo, contenido, o enlace. Define una Prioridad, para resolver los conflictos entre eventos. Los eventos pueden utilizarse con diferentes propósitos: modelado de elementos virtuales (nodos, contenidos, anclas, enlaces, atributos, eventos, o sus relaciones) que se crean en tiempo de ejecución; enlaces condicionales, cuyos destinos varíen en función de la situación; y cualquier tipo de comportamiento interactivo. La Figura 2.41 muestra un ejemplo de esta función.

```
el(Don Quijote – Dulcinea) = {Evento A, 2}
```

Figura 2.41 Ejemplo de asignación de un evento a un nodo

En este ejemplo se asigna el `Evento A` al nodo `Don Quijote – Dulcinea`, fijando su prioridad a `2`. El `Evento A` (definido en la Figura 2.38) se evalúa cuando se accede al `nodo Don Quijote – Dulcinea`.

LA FUNCIÓN DE CATEGORÍA DE ACCESO: $ac(U_i, x) = CategoríaAcceso_i$.

La *función de categoría de acceso* asigna un valor de capacidad de acceso al usuario para el hiperdocumento y cada nodo o contenido de la aplicación hipermedia. La Figura 2.42 muestra un ejemplo.

```
ac(Carmen, Don Quijote – Dulcinea) = personalización
ac(Carmen, Discusión sobre Don Quijote – Dulcinea) = presentación
ac(Mario, Ballet) = 0
```

Figura 2.42. Ejemplos de asignación de categorías de acceso

En el ejemplo a `Carmen` se le permite modificar el nodo `Don Quijote – Dulcinea`, y ver el contenido de `Discusión sobre Don Quijote`. `Mario` no puede visualizar el nodo `Ballet`, bajo ninguna circunstancia.

Como ya hemos comentado el modelo incluye 86 operaciones asociadas a los conjuntos y funciones anteriores cuya finalidad es la gestión del hiperdocumento. Las operaciones permiten crear, modificar y eliminar cualquiera de los elementos de los conjuntos y/o funciones anteriores. En [Díaz 01] puede encontrarse el listado completo.

Ejemplo de Labyrinth dinámico

El modelo permite especificar cualquier actividad computacional que deba ejecutarse dentro de la aplicación hipermedia. Con este fin se puede utilizar un lenguaje de propósito general que permite especificar la reacción computacional a cualquier evento, incluyendo la creación de contenidos y enlaces en tiempo de ejecución. Por ejemplo supongamos que debemos modelar una aplicación donde las palabras de un texto enlazan con un glosario. En principio los enlaces del texto con el glosario pueden modelarse de dos formas distintas: enlazando cada palabra del texto con cada definición del glosario; o estableciendo un único *vínculo virtual*. Los vínculos virtuales representan enlaces cuyo origen o destino depende de parámetros que se conocen en tiempo de ejecución, y por tanto son creados bajo demanda.

La solución para el segundo caso propuesta por Labyrinth se muestra en la Figura 2.43, la cual recoge parte de la notación gráfica utilizada por el modelo.

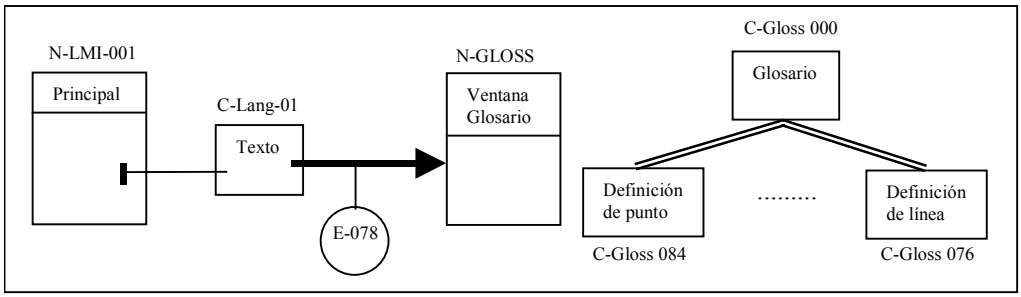

Figura 2.43 Representación gráfica del modelado de un glosario

En esta figura podemos observar los nodos para la pantalla principal (N-LMI-001), y para la ventana del glosario (N-GLOSS). También podemos ver el contenido texto (C-Lang-01), y el contenido glosario (C-Gloss 000), el cual está formado por agregación de otros contenidos (C-Gloss 084, ..., C-Glos-076). El contenido del texto está asociado a la pantalla principal. Además entre este contenido y la ventana del glosario existe un vínculo virtual con un evento asociado (E-078). Aunque el diagrama no lo muestra existen anclas que conectan el texto con las definiciones del glosario. Cuando se seleccionan, la especificación del ancla origen del enlace se utiliza para poner el contenido correcto en la ventana del glosario. Para optimizar el proceso de búsqueda, el glosario está formado por definiciones agregadas, caracterizadas por un identificador único. De esta forma en enlace glosario se especifica de manera virtual ligando un evento al enlace referencial entre el texto y la ventana del glosario. La Figura 2.44 recoge la especificación formal del evento.

```
IdEvento: E-078
Condiciones: seleccionar enlace
Acciones:
        key= copyChunk(basicHD, self.LinkStart.ContentId, self.LinkStart.AnchorPos.Position,
                        self.LinkStart.AnchorPos.Extension)
        glossaryTerms= getContentComponents(basicHD, C-Gloss-000)
        target= search(key, glossaryTerms)
        if target is not null then
                placeContentNode(basicHD, target.ContentId, N-Gloss, (0, 0))
                activateLink(basicHD, self.LinkId)
        endif
```

Figura 2.44 Especificación formal del evento E-078

2.6.1.2 Ejemplo

Veamos como representa Labyrinth el hipertexto del ejemplo. En cuanto a la parte estática de los contenidos Labyrinth no tiene ningún problema. Respecto a la parte dinámica, se necesitan un par de vínculos virtuales que mostraremos más adelante. Tenemos lo siguiente.

El conjunto de usuarios viene definido por:

```
{Autores, Grupo, [Antonio]}
{Antonio, Individual, [Autores]}
```

El conjunto de nodos viene definido por:

```
{nodoVentana1, personalización}
{nodoVentana2, personalización}
{nodoPanel2_1, personalización}
{nodoPanel2_2, personalización}
```

El conjunto de contenidos viene definido por:

```
{biografía, [texto, [palabra]]}, {españa, [foto, [pixel]]},
{madrid, [texto, [palabra]]}, {lepe, [texto, [palabra]]},
{estudios, [texto, [palabra]]}, {docencia, [texto, [palabra]]}
{isg1, [texto, [palabra]]}, {isg2, [texto, [palabra]]},
{isg91, [texto, [palabra]]}, {formulario, [texto, [palabra]]} }
```

Las anclas que son origen de enlaces referenciales son (obviamos su posición concreta):

```
{bioAEsp, biografía, _}, {espABio, españa, _},
{espAMad, españa, _}, {madAEsp, madrid, _},
{espALepe, españa, _}, {lepeAEsp, lepe, _},
{bioAEst, biografía, _}, {estABio, estudios, _},
{bioADoc, biografía, _}, {bioADoc1, biografía, _},
{bioADoc2, biografía, _}, {bioADoc91, biografía, _},
{docABio, docencia, _},
{docAIsg1, docencia, _}, {isg1ADoc, isg1, _},
{docAIsg2, docencia, _}, {isg2ADoc, isg2, _},
```

```
{docAIsg91, docencia, _}, {isg91ADoc, isg91, _},
{boton, formulario, _}
```

Nótese como ahora si vamos a ser capaces de capturar las relaciones que existen entre la `biografía` y la `docencia`. Las anclas que son destino de enlaces referenciales son:

```
{biografía, biografía, _},
{españa, españa, _},
{madrid, madrid, _}, {lepe, lepe, _},
{estudios, estudios, _},
{docencia, docencia, _}, {docencia1, docencia, _}, {docencia2, docencia, _},
{docencia91, docencia, _},
{isg1, isg1, _}, {isg2, isg2, _}, {isg91, isg91, _}
```

También necesitamos asociar anclas a los nodos que conforman la pantalla formada por las dos ventanas para poder representar los enlaces de agregación:

```
{nodoVentana2, nodoVentana2, _},
{nodoPanel2_1, nodoPanel2_1, _},
{nodoPanel2_2, nodoPanel2_2, _}
```

El conjunto de enlaces referenciales vienen dados por:

```
{eBioEsp, [bioAEsp], [españa]}, {eEspBio, [espABio], [biografía]}, {eEspMad, [espAMad],
[madrid]}, {eMadEsp, [madAEsp], [españa]}, {eEspLepe, [espALepe], [lepe]}, {eLepeEsp,
[lepeAEsp], [españa]}, {eBioEst, [bioAEst], [estudios]}, {eEstBio, [estABio],
[biografía]}, {eBioDoc, [bioADoc], [docencia]}, {eBioDoc1, [bioADoc1], [docencia1]},
{eBioDoc2, [bioADoc2], [docencia2]}, {eBioDoc91, [bioADoc91], [docencia91]},
{eDocBio, [docABio], [biografía]},
{eDocIsg1, [docAIsg1], [isg1]}, {eIsg1Doc, [isg1ADoc], [docencia]},
{eDocIsg2, [docAIsg2], [isg2]}, {eIsg2Doc, [isg2ADoc], [docencia]},
{eDocIsg91, [docAIsg91], [isg91]}, {eIsg91Doc, [isg91ADoc], [docencia]}
```

Nótese como ahora si hemos podido representar los múltiples enlaces de `biografía` con `docencia`. También necesitamos un enlace que determine la agregación:

```
{ePant2V2_1V2_2, [nodoVentana2], [nodoPanel2_1, nodoPanel2_2] }
```

Las definiciones de atributos van a fijar el tipo de enlace a `referencial`, y la dirección de los mismos a `unidireccional` (nótese que no hay ningún error, pues al ser distintas anclas las origen que las destino los enlaces no pueden ser bidireccionales):

```
{TipoEnlace, "Referencial"}
{Dirección, "Unidireccional"}
```

Ahora definiremos el evento para la selección de enlaces referenciales:

```
{Evento selecciónAncla, activarEnlace, [código evento que muestra el destino de
un enlace cuando se selecciona el ancla origen]}
```

También necesitamos un evento para la búsqueda de resultados a partir del formulario, y otro para responder a los enlaces de los contenidos con los que enlace dicho resultado de búsqueda.

```
{Evento botónFormulario, pulsarBotón, [código evento que muestra el resultado de
una búsqueda en el formulario]}

{Evento selecciónAnclaResultados, activarEnlaceResultados, [código evento que
muestra el destino de un enlace cuando se selecciona el ancla origen en el
resultado de la búsqueda del formulario]}
```

Ahora asignaremos contenidos a los nodos. Como nos preocupan las restricciones espaciales o temporales simplificaremos la función `lo`. En efecto:

```
lo(biografía, nodoVentana1) = {_, _}
lo(españa, nodoPanel2_1) = {_, _}
lo(madrid, nodoPanel2_2) ) = {_, _}
lo(lepe, nodoPanel2_2) ) = {_, _}
lo(estudios, nodoPanel2_1) ) = {_, _}
lo(docencia, nodoPanel2_1) ) = {_, _}
lo(isg1, nodoPanel2_2) ) = {_, _}
lo(isg2, nodoPanel2_2) ) = {_, _}
lo(isg3, nodoPanel2_2) ) = {_, _}
lo(formulario, nodoPanel2_1) = {_, _}
```

Ahora deberíamos fijar los valores de los atributos para los elementos. Obviaremos a los que toman sus valores de la definición por defecto, mostrando solo el enlace de agregación:

```
al(ePant2V2_1V2_2) = {(Etiqueta, "Composición"), (Autor, "Antonio"),
                      (TipoEnlace, "Agregación"), (Dirección, Valor por defecto) }
```

También debemos asignar cada el evento a los enlaces:

```
el(eBioEsp) = {Evento selecciónAncla, 1}
el(eEspBio) = {Evento selecciónAncla, 1}
el(eEspMad) = {Evento selecciónAncla, 1}
el(eMadEsp) = {Evento selecciónAncla, 1}
el(eEspLepe) = {Evento selecciónAncla, 1}
el(eLepeEsp) = {Evento selecciónAncla, 1}
el(eBioEst) = {Evento selecciónAncla, 1}
el(eEstBio) = {Evento selecciónAncla, 1}
el(eBioDoc) = {Evento selecciónAncla, 1}
el(eBioDoc1) = {Evento selecciónAncla, 1}
el(eBioDoc2) = {Evento selecciónAncla, 1}
el(eBioDoc91) = {Evento selecciónAncla, 1}
el(eDocBio) = {Evento selecciónAncla, 1}
el(eDocIsg1) = {Evento selecciónAncla, 1}
el(eIsg1Doc) = {Evento selecciónAncla, 1}
el(eDocIsg2) = {Evento selecciónAncla, 1}
el(eIsg2Doc) = {Evento selecciónAncla, 1}
el(eDocIsg91) = {Evento selecciónAncla, 1}
el(eIsg91Doc) = {Evento selecciónAncla, 1}
```

También tenemos que asignar eventos a los nodos para responder al comportamiento dinámico.

```
el (formulario) = {Evento botónFormulario, 1}
el (nodPanel2_2) = {Evento seleccionAnclaResultados, 1}
```

Finalmente tenemos que asignar los permisos de acceso. Lo simplificaremos con la siguiente expresión:

```
ac(Antonio, e) = total, pt e ∈ H
```

La representación gráfica (simplificada de este hipertexto) viene recogida en la Figura 2.45, donde el evento `selecciónAncla` se muestra como genérico para todos los enlaces referenciales que no tienen asignado ninguno.

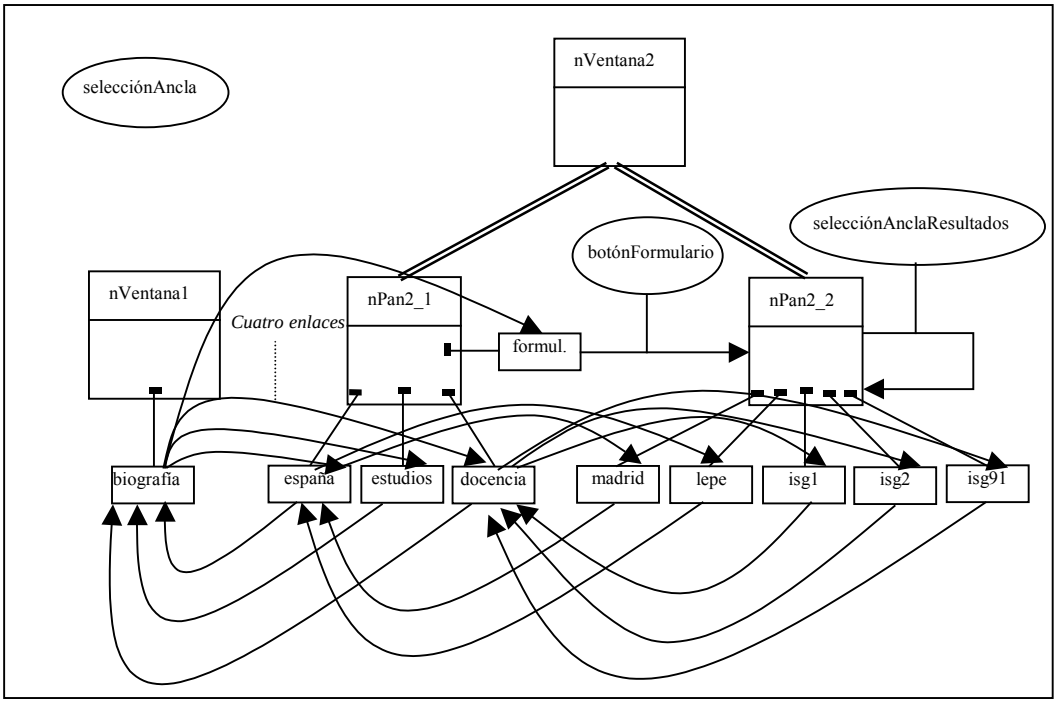

Figura 2.45 Representación gráfica del hipertexto

Nótese como los enlaces referenciales virtuales quedan caracterizados por los eventos `botónFormulario` (lo representamos asignado al enlace virtual), y `selecciónAnclaResultados`. En cuanto a la presentación concreta de los contenidos durante la ejecución de la aplicación, el modelo recurre a atributos asignados a los nodos que actúan como las especificaciones de presentación del modelo Dexter. Hemos obviado estos atributos en aras de una mayor simplicidad.

2.6.1.3 Discusión y comparativa

El modelo Labyrinth, al igual que OOHDM permite expresar cualquier actividad computacional que deba llevarse a cabo dentro de la aplicación hipermedia. Con este fin recurre al concepto de evento. Un evento es el código de una expresión computacional en un lenguaje de propósito general. Por tanto presenta la ventaja de poder especificar cualquier actividad computacional, pero paga el precio de tener que recurrir a una especificación completa mediante eventos de cual va a ser el comportamiento dinámico de la aplicación. Esto se debe a que a diferencia del modelo Amsterdam, Trellis o de hipergrafos, y al igual que el OOHDM, no fija ninguna semántica de navegación por defecto, ya que su genericidad se lo impide.

Separación de contenidos, navegación y presentación. El modelo presenta una clara separación del nivel de composición interna frente al de almacenamiento. Ahora el nivel de almacenamiento es capaz de recoger cualquier tipo de relación que se establezca entre los contenidos del nivel de composición interna. La función de localización de contenidos, permite asignar contenidos a los nodos, y fijar la presentación de estos. El nivel de presentación se basa en la función anterior, y en los eventos asociados tanto a nodos como a enlaces y contenidos.

Contexto. El modelo Labyrinth no recoge directamente la idea de contexto (como hace el modelo Amsterdam). Al igual que OOHDM se apoya en la potencia expresiva de los eventos (OOHDM se apoya en ADVs, o *Abstract Data View*) para expresar la noción de contexto, junto con atributos asignados a los nodos. De esta forma se deben especificar eventos para indicar cual va a ser el comportamiento dinámico de los nodos respecto a la interfaz.

Sincronización. Al igual que el modelo Amsterdam incluye un soporte explícito para la sincronización. Difiere del modelo OOHDM en que ahora las primitivas están incorporadas en el modelo, y no es necesario recurrir a ADVs u otros formalismos para expresar la sincronización entre elementos del modelo.

Formalización de la semántica de navegación. A pesar de una completa formalización, el modelo no incluye un formalismo explícito para representar el esquema navegacional de la aplicación, más allá de la información proporcionada por las relaciones entre nodos (como las de agregación en la Figura 2.45). Incluye un formalismo gráfico que ayuda bastante a entender la especificación funcional/textual del hipertexto. Por otro lado la semántica de navegación se determina en base a la función de localización y los eventos definidos para cada aplicación. Es por tanto imposible proporcionar una semántica de navegación genérica, y por consiguiente resulta más costosa la generación de prototipos/aplicaciones que frente a otros modelos que restringen la semántica de navegación (como Amsterdam, o Trellis, por ejemplo).

Organización relacional de los contenidos. El modelo Labyrinth no supone ninguna organización de los contenidos. Si se determina parcialmente la estructura interna de estos con el fin de poder asignar anclas a los mismos.

Gestión explícita de eventos. Como ya hemos comentado Labyrinth incluye una gestión explícita de eventos. Estos pueden especificarse informalmente en lenguaje natural, o con total formalidad en un lenguaje de propósito general. Si se utiliza la segunda aproximación aparece una divergencia con OOHDM, ya que este utiliza diagramas para especificar dicho código en lugar de especificar el propio código. Esta diferencia se debe a que Labyrinth es un modelo hipermedia, mientras que OOHDM es una metodología de diseño hipermedia.

Generación dinámica de contenidos. El modelo incluye la posibilidad de incluir contenidos y enlaces generados dinámicamente a través de los enlaces virtuales y eventos asociados a dichos enlaces. El inconveniente de esta aproximación es la imposiblidad de proporcionar una semántica de navegación por defecto. El modelo propuesto en esta tesis (Capítulo 5) aúna la posibilidad de generar enlaces y contenidos de manera dinámica, y la presencia de una semántica de navegación por defecto. Esto es posible gracias a la presencia de una *función de generación de contenidos* y a una semántica de navegación establecida sobre una *función de relación*, la cual a su vez puede definirse extensionalmente para contenidos dinámicos, e intensionalmente (en base a la función de generación de contenidos) para contenidos dinámicos.

Gestión de usuarios y seguridad. Este modelo es el único sistema de representación hipermedia que formaliza totalmente la gestión de usuarios y de seguridad.

Distinción de enlaces a nivel contenidos y esquema navegacional. El modelo no hace una distinción explícita entre los enlaces a nivel de contenidos y a nivel navegacional. Resulta muy interesante la posibilidad de expresar enlaces virtuales (como el descrito en [Díaz 01]), donde solo se utilizan las anclas, y los enlaces se calculan dinámicamente mediante un evento asociado al enlace referencial. Como ya hemos comentado, presenta el inconveniente de tener que proporcionar el evento asociado, dificultando así la legibilidad del esquema navegacional.

Técnicas orientadas a objetos. El modelo no utiliza explícitamente técnicas orientadas a objetos para expresar los eventos, aunque estos si que podrían expresarse en un lenguaje orientado a objetos. De todas formas, no habría ningún problema en proporcionar diagramas orientados a objetos como representación del código de los eventos.

2.6.2 La metodología de diseño hipermedia orientada a objetos (OOHDM)

2.6.2.1 Presentación

Introducción

La metodología de diseño hipermedia orientada a objetos, OOHDM (del inglés, *Object-Oriented Hypermedia Design Methodology*) [Schwabe 95a], [Schwabe 96], es una metodología de diseño y construcción de aplicaciones hipermedia basada en el modelo orientado a objetos. En el Capítulo 3 estudiaremos la metodología de diseño y construcción, centrándonos ahora en las técnicas de modelado utilizadas. OOHDM consta de tres vistas o esquemas: el *esquema conceptual*, el *esquema navegacional*, y el *esquema de la interfaz abstracta*. El primero se encarga de identificar los elementos del dominio y las relaciones existentes entre estos. El segundo se encarga de identificar las relaciones navegacionales existentes entre estos elementos. El tercero se encarga de

proporcionar diferentes interfaces a estas relaciones navegacionales. Es una metodología extremadamente potente capaz de representar (desde el punto de vista del análisis y diseño) cualquier actividad computacional.

Esquema conceptual

El *esquema conceptual* proporciona un modelo del dominio de la aplicación basado en modelos de análisis y diseño orientados a objetos ([Rumbaugh 97] en el modelo original y [Booch 99] en las últimas evoluciones), aumentado con algunas primitivas propias. Este diseño utiliza (representaciones orientadas a objetos de) clases para modelar los elementos del dominio (RMDM utilizaba entidades, por ejemplo). Estas clases pueden construirse utilizando los mecanismos orientados a objetos de agregación y de generalización/especialización. También aparecen relaciones genéricas entre dichas clases. Los diagramas obtenidos en esta vista deben capturar las relaciones existentes entre los elementos de contenidos de la aplicación hipermedia con independencia del tratamiento posterior que se quiera dar a estos contenidos y/o a sus relaciones. La Figura 2.46 muestra un ejemplo de parte de una vista conceptual de una aplicación hipermedia.

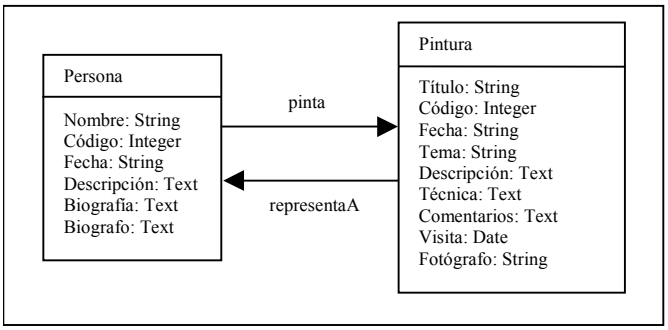

Figura 2.46 Parte de la vista conceptual

En esta figura podemos observar dos clases, una para Personas, y otra para Pinturas. Entre ambas existen dos relaciones, pinta y representaA.

Esquema navegacional

En OOHDM una aplicación se concibe como una *vista navegacional* sobre el esquema conceptual. El esquema navegacional se expresa en dos esquemas: *el esquema de clases navegacionales*, y el *esquema de contexto de navegación*. Los objetos susceptibles de aparecer en la vista navegacional se definen en el esquema de clases navegacionales, cuyas clases reflejan la vista elegida sobre el dominio de aplicación. El modelo proporciona un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso.

Los *nodos* son los contenedores básicos de información en las aplicaciones hipermedia. Se definen como vistas orientadas a objetos de las clases conceptuales utilizando un lenguaje de interrogación (del inglés *query*). Este lenguaje permite definir a un nodo como combinación de atributos de distintas clases de la vista conceptual. Contiene atributos de tipos simple, y anclas de enlaces. La Figura 2.47 muestra un nodo.

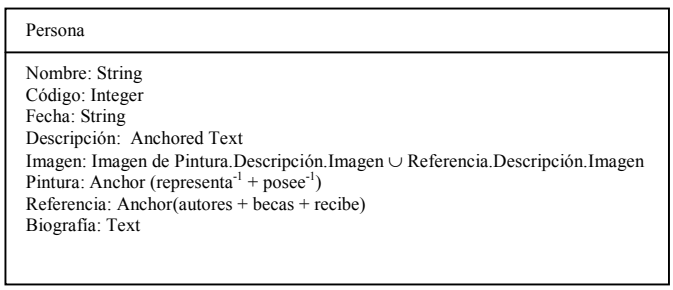

Figura 2.47 La clase navegacional nodo para Persona

En esta figura podemos observar como la clase navegacional prescinde de ciertos atributos (Biógrafo) e incluye otros nuevos, seleccionados de otras clases del dominio (Imagen). También incluyen información sobre anclas.

Los *enlaces* reflejan relaciones que serán exploradas por el usuario final. En el modelo, los enlaces implementan las relaciones definidas en el esquema conceptual. Es decir, los enlaces son las realizaciones navegacionales de las relaciones conceptuales. Se pueden definir clases de enlaces para especificar atributos de enlaces (propiedades del enlace) y comportamiento, objetos origen y destino y cardinalidad.

Las *estructuras de acceso* actúan como índices o diccionarios y son útiles para ayudar al usuario en la búsqueda de la información final. Menús, índices y tours guiados son ejemplos de estas estructuras de acceso.

El modelo considera como la principal estructura del esquema navegacional al *contexto navegacional*. Un contexto navegacional es un conjunto de nodos, enlaces, clases de contextos y otros contextos navegacionales anidados. Los contextos indican que clases navegacionales (o nodos) son seleccionables. Es decir, recogen parcialmente la noción de contexto del modelo Amsterdam. Para capturarla totalmente OOHDM utiliza además mapas navegacionales y mapas de ADVs. La Figura 2.48 muestra un ejemplo de contexto.

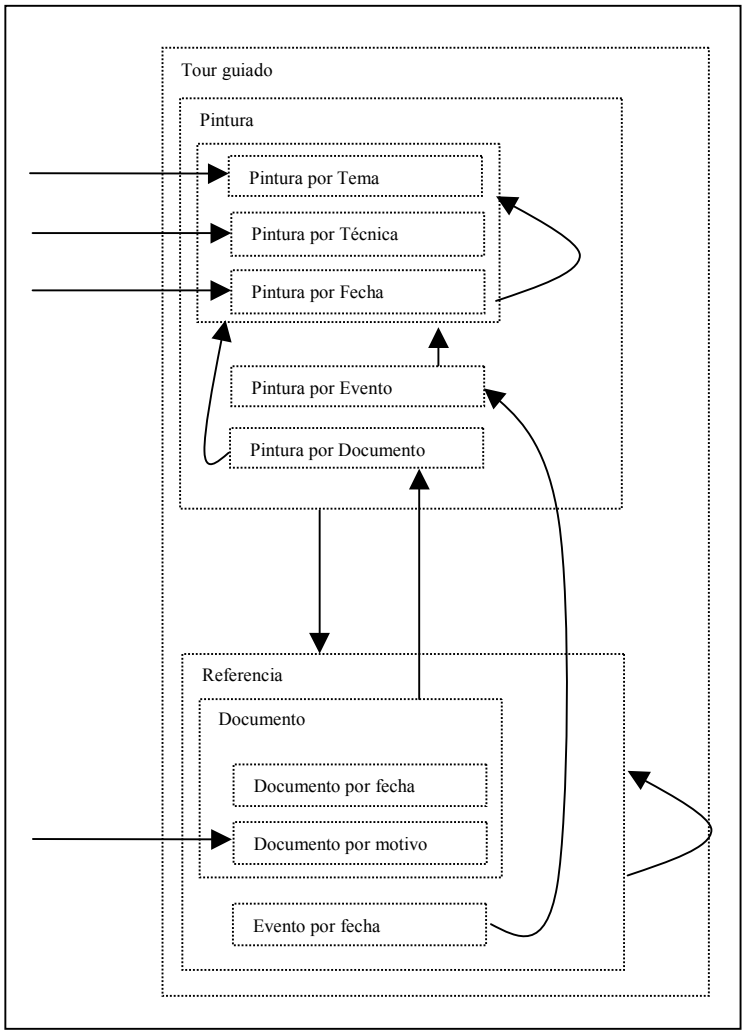

Figura 2.48 Ejemplo de contexto

Pintura por <evento>
incluye: ∀ a ∈ Pintura, e ∈ Evento | (p, a) ∈ exhibido ∧ e.Titulo = <evento>
puntos de entrada: e1 [primer nodo]
camino: circular, ordenado ascendente sobre p.Fecha, P ∈ Pintura
conportamiento: paso a paso

Figura 2.49 Definición del contexto navegacional Pintura por Evento

En esta figura podemos apreciar un contexto para un Tour guiado. Si suponemos que accedemos al contexto por el nodo de Pintura por Tema somos capaces de seguir viendo Pinturas por técnicas, o Pinturas por fechas. Desde este contexto también está seleccionable el contexto de Referencia. Si pasamos a este contexto podemos ver Documentos por fecha, o Documentos por motivo. También podemos ver Eventos por fecha. Si estamos dentro de este último contexto, podremos pasar al contexto de ver Pinturas por evento, y una vez aquí podemos seguir viendo las pinturas por Tema, Técnica, o Fecha. La Figura 2.49 muestra la definición de uno de los contextos que aparece en el esquema de contexto anterior.

Este contexto indica que se deben mostrar las pinturas correspondientes a un cierto evento (por ejemplo "El descubrimiento de América"), y cierta información sobre como van a ser visualizadas. Las *clases de contexto* complementan la definición de clases navegacionales (nodos) indicando que información se muestra y que anclas están disponibles cuando se accede a un objeto en un contexto particular. La Figura 2.50 muestra la clase de contexto para los objetos que van a ser mostrados en el contexto anterior.

Pintura en evento
Previo: Ancla (previa) Siguiente: Ancla (siguiente)
Ámbito: Pintura por evento Parte-de: Pintura

Figura 2.50 Clase de contexto para el contexto anterior

El comportamiento dinámico de la estructura navegacional se especifica completamente mediante la definición de transformaciones navegacionales que ocurren mientras se activan enlaces. En algunos casos, por ejemplo, se puede desear especificar que el nodo fuente permanece activo, y que el nodo destino se mantiene también activo; o que todos los nodos destino de un enlace n-ario se activan simultáneamente. El modelo utiliza transiciones de estados orientadas a objetos derivados de los mapas de estados (del inglés *Statecharts*), y que denomina, *mapas de navegación*. Estas cartas soportan anidamiento estructural y de comportamiento, y permiten expresar el comportamiento dinámico. El mapa de navegación se complementa con un conjunto de predicados expresados en Lógica Temporal que permiten validar la especificación contra propiedades como seguridad, persistencia, etc. La Figura 2.51 muestra un mapa navegacional.
En este ejemplo cuando se navega de Persona a Referencia los nodos permanecen activos (eso indica el círculo delante de la flecha). Sin embargo cuando pasamos de Pintura a Evento, todos los nodos de la agrupación se cierran. Las líneas discontinuas indican anidamiento aditivo.

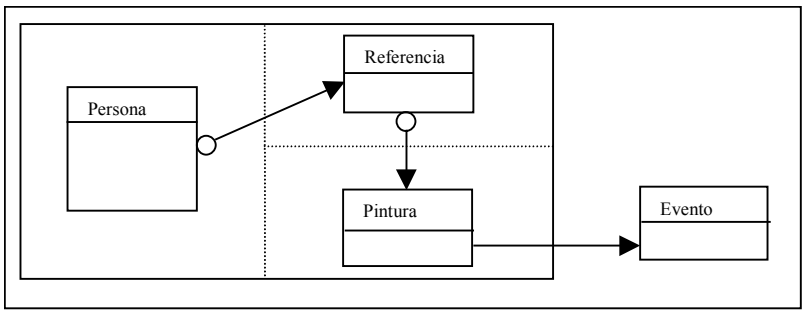

Figura 2.51. Un ejemplo de mapa navegacional

Esquema de interfaz abstracta

Una vez definida la estructura navegacional, debe hacerse perceptible al usuario a través de la interfaz de la aplicación, especificada a través del *modelo de interfaz abstracta*. Esto significa definir que objetos de interfaz percibirá el usuario, que aspecto tendrán los objetos navegacionales, el modo de sincronización de los objetos multimedia, y que transformaciones de interfaz sucederán. Una separación clara entre el diseño navegacional y el de interfaz abstracta, permite construir distintas interfaces para el mismo modelo navegacional. El modelo utiliza *vistas de datos abstractas*, ADV (del inglés, *Abstract Data View*) para describir la interfaz de usuario de una aplicación hipermedia.

Las *vistas abstractas de usuario* son modelos formales de interfaces de objetos que se especifican mostrando:
- El modo en que se construyen. Las propiedades de percepción también se especifican como atributos o partes de una ADV.
- El modo en que se relacionan con los objetos navegacionales. OOHDM utiliza *Diagramas de configuración* con este fin.
- Como se comportan frente a eventos externos; en particular como activan la navegación y que transformaciones de interfaz ocurren cuando el usuario interactúa con la aplicación.

Las ADVs son objetos que poseen un estado y una interfaz, donde la interfaz puede ejercitarse a través de funciones o llamadas a procedimientos o eventos de entrada y salida. Las ADVs son abstractas en la medida de que solo representan la interfaz y el estado, y no la implementación. En una aplicación típica que utilice ADVs, hay un conjunto de ADOs (*objetos abstractos de datos*) que manejan estructuras de datos y control dentro de la aplicación, y un conjunto de objetos de interfaz (instancias de ADVs) manejando aspectos de la interfaz de la aplicación, tales como, la entrada del usuario al sistema, y la salida de este. En el contexto de OOHDM, los objetos navegacionales como nodos, enlaces o estructuras de acceso actúan como ADOs, y su ADV asociada se utiliza para especificar su apariencia al usuario.

Cuando se utiliza una ADV en el diseño de una aplicación hipermedia, puede entenderse como un objeto de interfaz que compromete un conjunto de atributos, los cuales definen sus propiedades de percepción, y el conjunto de eventos que pueden manejar, (por ejemplo los generados por el usuario). Los valores de atributos se pueden definir como constantes definiendo un estilo particular de apariencia tales como posición, color, o sonido. Una variable reservada, percepciónDeContexto, se utiliza para indicar modificaciones al estado de percepción. Cuando se desea hacer a algún objeto perceptible, se añade a la percepciónDeContexto, y los elementos eliminados de la percepciónDeContexto ya no son perceptibles.

Definir un modelo de interfaz abstracta de una aplicación hipermedia en términos de ADVs implica:

- Definir ADVs para cada objeto navegacional. De hecho, definir al menos un ADV para cada clase navegacional.
- Especificar el diagrama de configuración mostrando las relaciones estáticas entre ADVs y ADOs (que son los objetos navegacionales).
- Especificar el mapa de ADV para cada ADV que muestre el comportamiento dinámico de la aplicación.

Los *diagramas de configuración* son útiles para expresar patrones de comunicación entre objetos, en términos de servicios proporcionados y requeridos. En la aproximación de diseño ADV se utilizan para representar eventos externos (iniciados por el usuario) que maneja una ADV; los servicios proporcionados por la ADV (como mostrar) y la comunicación entre ADVs y ADOs. Un diagrama de configuración podría indicar también la estructura anidada de ADVs. En el contexto hipermedia, interesa definir el modelo en que el usuario interactuará con la aplicación hipermedia, y en particular con los objetos de interfaz que causarán la navegación. Cada clase nodo definirá una interfaz pública con los servicios proporcionados por sus objetos. En particular todos los nodos reaccionaran al mensaje: anclaSeleccionada(A) iniciando la navegación a través del enlace asociado al ancla A. En una ADV compuesta por diferentes objetos de interfaz como botones asociados con nodos ancla, se anota el nombre del ancla en el diagrama de configuración. La Figura 2.52 muestra un diagrama de configuración.

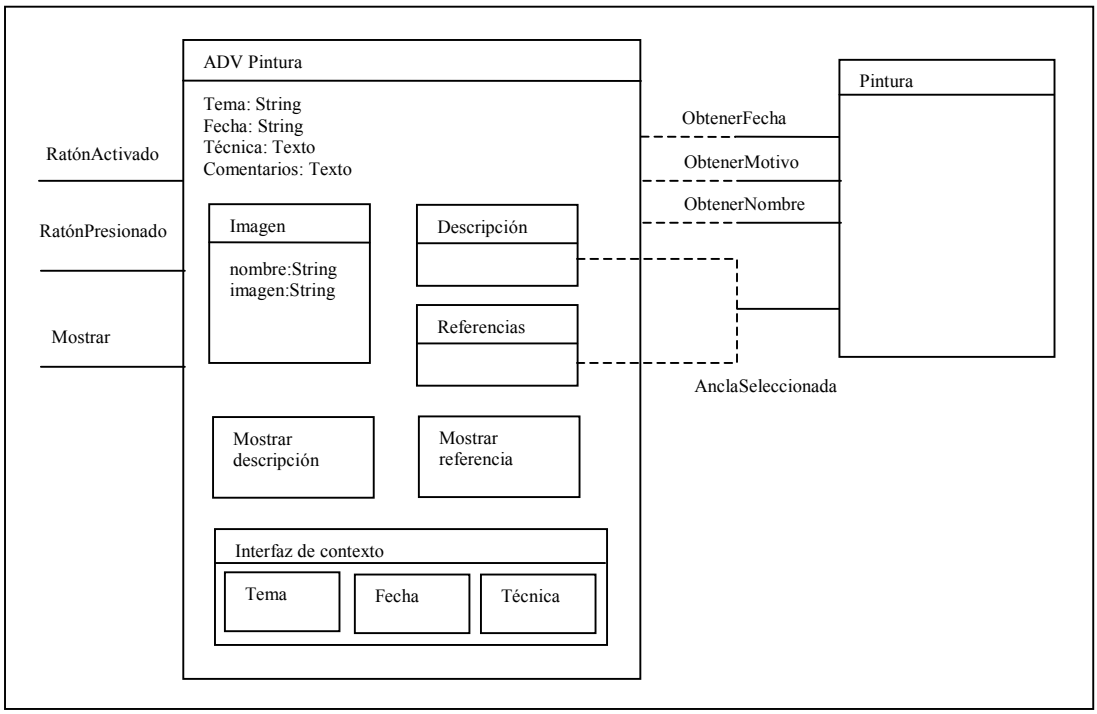

Figura 2.52 Diagrama de configuración

En esta figura se indica que una pintura puede reaccionar a los eventos externos: RatónActivado, RatónPresionado, Mostrar, y comunicarse con el ADO que lo posee (el nodo Pintura) enviando los mensajes: anclaSeleccionda, obtenerFecha, obtenerMotivo, obtenerNombre.

El modelo utiliza *mapas ADV*, otra notación derivada de los mapas de estados, que añaden anidamiento estructural y de comportamiento, y una notación similar a la de las redes de Petri para expresar sincronización cuando se manejan datos multimedia. La Figura 2.53 muestra un mapa ADV simplificado. Entre otras cosas especifica que cuando un usuario selecciona un ancla en Referencias, se activa el correspondiente ADV.

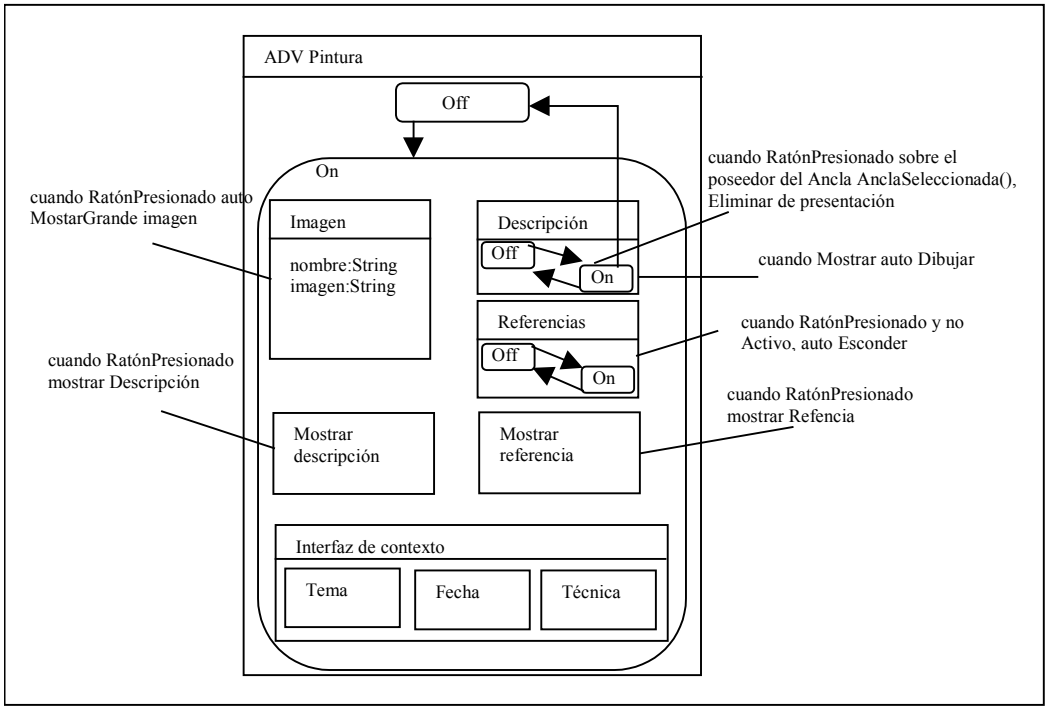

Figura 2.53 Mapa ADV

2.6.2.2 Ejemplo

El modelo OOHDM es lo suficientemente potente como para representar el ejemplo que venimos manejando, incluido los contenidos generados dinámicamente. Aunque presenta el mismo problema que los modelos relacionales. ¿Cómo deberíamos definir el esquema conceptual?, es decir, ¿qué es lo que procede? Considerar que todas las entidades que aparecen en el ejemplo son objetos de la misma clase (algo similar a lo propuesto en la Figura 2.26) o dividir los objetos en entidades según aparezcan en diferentes ventanas (algo similar a lo propuesto en la Figura 2.30). La primera opción es la correcta desde el punto de vista de modelado orientado a objetos, ya que desde este punto de vista ¿qué diferencia hay entre una biografía y la docencia?. Optaremos por una solución intermedia en la cual agruparemos entidades en base a su "tipo" más o menos bien definido. La Figura 2.54 ilustra esta solución.

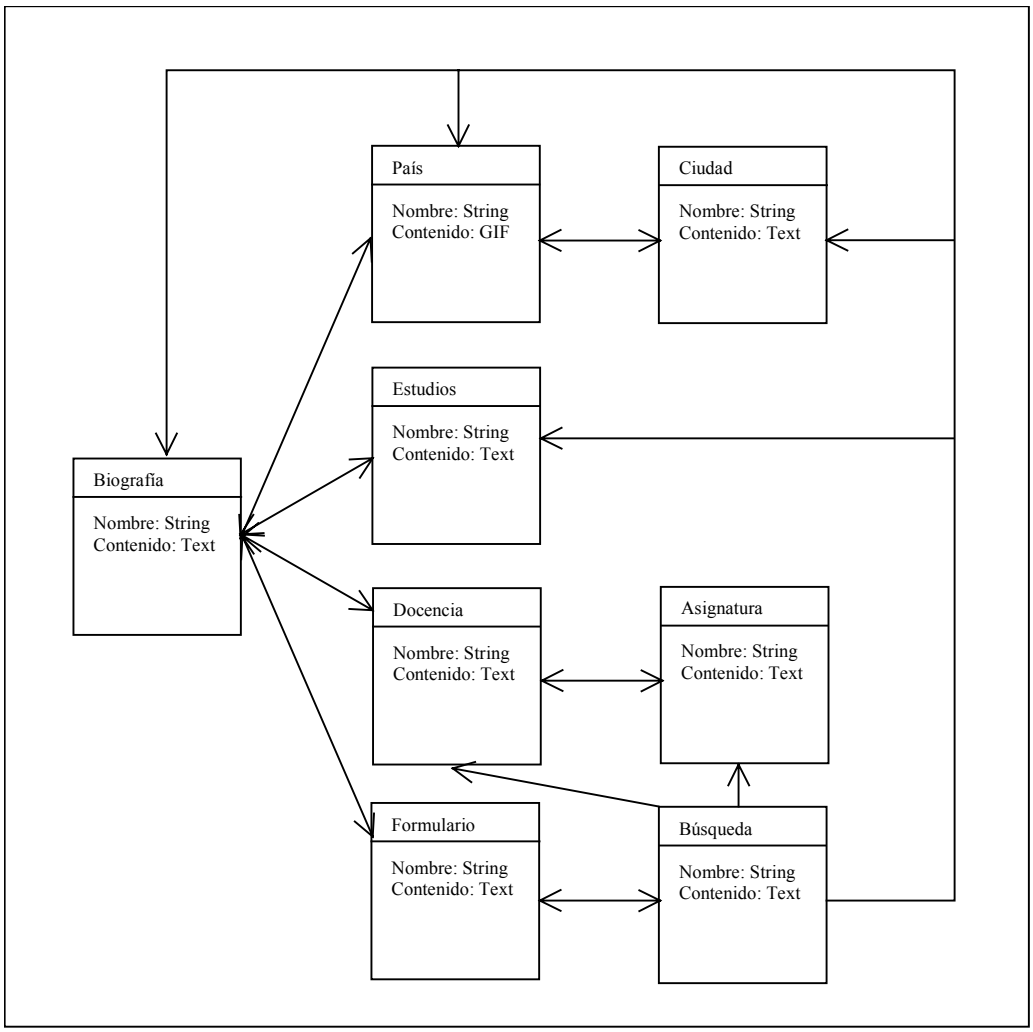

Figura 2.54 Esquema conceptual del ejemplo

En cuanto al esquema navegacional, las clases navegacionales (nodos) se corresponden con las anteriores clases, aumentándolas con información sobre las anclas. El esquema de contexto de navegación debe tener en cuenta que objetos son seleccionables desde otros objetos. Como el esquema navegacional habla de dos pantallas tenemos la Figura 2.55. Como hemos diseñado el esquema conceptual en base al navegacional, ambos esquemas son bastante similares.

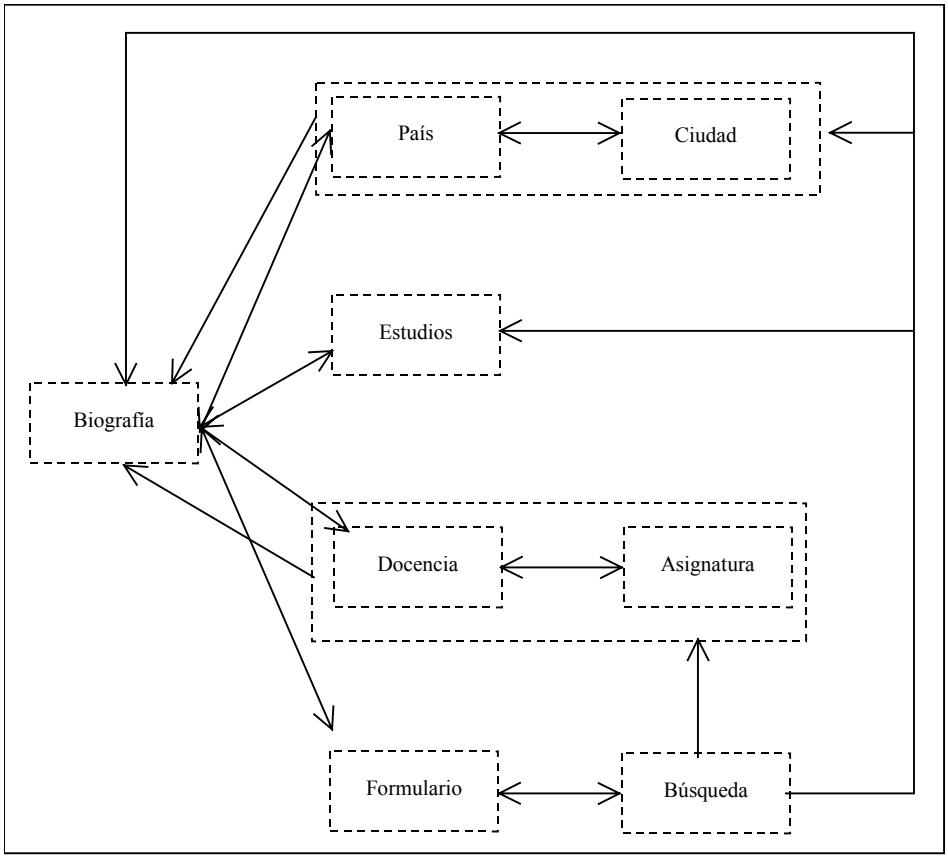

Figura 2.55 Esquema navegacional del ejemplo

Ahora deberíamos especificar las clases de contexto para indicar cual es el comportamiento de los contenidos y sus anclas cuando se accede a las clases navegacionales. Tambíen deberíamos especificar las transformaciones navegacionales que ocurren cuando se recorren los enlaces, es decir, el concepto de *frame*. La Figura 2.56 muestra este mapa navegacional.

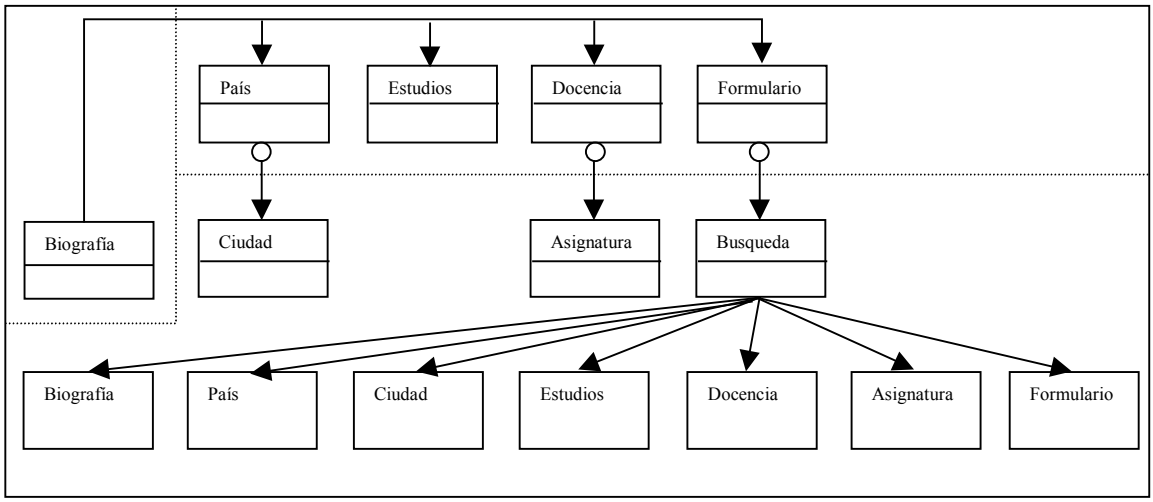

Figura 2.56 Mapa navegacional del ejemplo

Finalmente deberíamos proporcionar los ADVs para estas clases, no siendo extremadamente complejos, ya que los único que deben hacer las clases navegacionales es mostrarse cuando son referenciadas, y responder a la activación de las anclas, y a la actualización del resto de las clases. Como podemos ver este modelo es lo suficientemente potente como para representar "casi cualquier cosa", pero cuando las relaciones entre los

objetos del dominio no son fácilmente clasificable, como en este ejemplo, el modelo se encuentra con problemas similares a los modelos relacionales.

2.6.2.3 Discusión y comparativa

El modelo de datos proporcionado por la metodología OOHDM es con mucho el sistema de representación más potente visto hasta ahora. Se basa en no suponer ninguna característica implícita o por defecto de los componentes que forman una aplicación hipermedia, y por tanto es capaz de especificar (mediante esquemas de diseño) cualquier actividad computacional. La aproximación es similar (desde el punto de vista de implementación) a utilizar una herramienta de construcción de aplicaciones hipermedia (por ejemplo Authorware) o programar directamente la aplicación utilizando un lenguaje de propósito general (por ejemplo Java). Si utilizamos Authorware nos ahorraremos gran parte del trabajo, ya que muchas cosas se suponen por defecto, pero tenemos el inconveniente de no ser capaces de especificar una actividad computacional de carácter genérico dentro de la aplicación hipermedia. Si utilizamos Java seremos capaces de especificar cualquier actividad computacional que deba llevar a cabo la aplicación hipermedia, pero también debemos construir desde cero (bueno, desde la API Swing) la aplicación. Esta metáfora aplicada al modelo OOHDM nos muestra que si bien es el sistema de representación más potente de todos paga el precio de una excesiva complicación, ya que para expresar una única pantalla, como la del ejemplo que venimos manejando, se necesita proporcionar: (i) un esquema conceptual, (ii) las clases navegacionales, (iii) el esquema de contexto, (iv) las definiciones de contexto para cada contexto, (v) las clases de contexto para cada clase navegacional, (vi) el mapa navegacional para el esquema navegacional, (vii) los diagramas de configuración entre ADVs y clases, y (viii) los ADVs para las clases. Como podemos observar la potencia tiene un precio.

Separación de contenidos, navegación y presentación. OOHDM parte de una escrupulosa separación entre estos tres niveles. Ahora el nivel de composición interna viene definido por el esquema conceptual. Este esquema se encarga de organizar los elementos del dominio en base a sus relaciones. Nótese que este nivel también podría considerarse, en cierta manera como el nivel de almacenamiento del modelo Dexter. El nivel de navegación viene definido por las clases navegacionales, el esquema de contexto, y el mapa navegacional. Las primeras traducen las clases del nivel de composición interna al de navegación. El esquema de contexto indica que clases están accesibles en cada momento de navegación. El último indica las transformaciones que deben realizarse a nivel navegacional cuando se activan los enlaces. Finalmente el nivel de presentación viene determinado por la información del nivel anterior, junto con las ADVs, que especifican el comportamiento a nivel interfaz de las clases navegacionales.

Contexto. La potencia expresiva permite representar cualquier tipo de contexto. En particular la noción de contexto se reparte entre el esquema de contexto (junto a las clases de contexto), el mapa navegacional, y las ADVs. En principio, cabe preguntarse como es posible que una noción de contexto, resoluble directamente a través de los componentes compuestos del modelo Amsterdam, o mediante la estructura de hipergrafos, o de red de Petri (con mayor o menor fortuna), ahora necesita de distintos mecanismos representacionales para indicarlo. La respuesta es sencilla. Los sistemas de representación anteriores suponían en mayor o menor medida una semántica de navegación por defecto asociada a las estructuras navegacionales, y por tanto solo había que configurar dichas estructuras convenientemente. Ahora OOHDM no supone ninguna semántica de navegación por defecto, teniéndose que especificar todo el comportamiento de la aplicación hipermedia desde casi cero. Esto presenta la ventaja de poder representar cualquier cosa, pero al precio de un mayor esfuerzo para especificar nociones básicas en sistemas hipermedia.

Sincronización. El modelo proporciona soporte total para la sincronización. No hay más que utilizar convenientemente los mapas ADVs para indicar las relaciones espacio temporales que existen entre los objetos (sus clases) de la aplicación. Al igual que la noción de contexto, estas especificaciones, en principio, parecen más complejas que las proporcionadas por el modelo Amsterdam.

Formalización semántica de navegación. El esquema navegacional viene representado por el esquema de contexto, las clases de contexto, y el mapa navegacional. En principio, aunque estructuras totalmente formales no permiten una generación automática de aplicaciones, ya que son simplemente esquemas de diseño. El problema es similar a lo que sucede con UML y código real.

Organización relacional de los contenidos. OOHDM opta por una representación orientada a objetos de los contenidos, muy similar como hemos visto a la relacional. Al igual que el modelo RMDM utiliza las

relaciones existentes entre clases a nivel conceptual para definir enlaces a nivel navegacional. Además para liberar el peso que inducen las clases conceptuales sobre la navegación y sobre los objetos concretos, el modelo introduce la noción de clase navegacional, en principio distinta de la clase conceptual, aunque en la práctica íntimamente relacionadas.

Gestión explícita de eventos. OOHDM permite tratar cualquier evento gracias a la potencia que proporciona la programación orientada a objetos. En efecto, ahora en el nivel de composición interna se puede especificar cualquier actividad computacional (gracias al concepto de clase conceptual). La representación navegacional de estas clases conceptuales (las clases navegacionales), junto con la potencia de los mapas ADVs permiten representar al modelo la gestión de cualquier evento.

Generación dinámica de contenidos. La naturaleza de notación de diseño le permite representar contenidos generados dinámicamente sin demasiadas dificultades. El problema es que no proporciona semántica de presentación por defecto, haciendo imposible (o extremadamente costoso), por ejemplo, la generación automática de prototipos y/o aplicaciones.

Gestión de usuarios y seguridad. OOHDM es una metodología con una notación diseño, y por lo tanto no tiene en cuenta ninguna característica sobre usuarios y seguridad. Si se deseasen especificar estas características, no sobre el modelo, sino sobre las aplicaciones descritas por él, bastaría con incorporar estas características a la maquinaria de clases.

Distinción de enlaces a nivel contenidos y esquema navegacional. OOHDM parte de una casi total separación de enlaces entre contenidos y el esquema navegacional. De esta forma, y aunque los enlaces navegacionales se crean a partir de las relaciones existentes a nivel conceptual, el esquema navegacional es totalmente independiente del esquema conceptual. Esto se consigue en dos niveles. El primero transforma las clases conceptuales en clases navegacionales, añadiendo o eliminando información, según convenga. El segundo se logra a través del esquema de contexto, las clases de contexto, y el mapa navegacional. Por otro lado, al igual que RMDM y HDM, presenta la gran ventaja de representar el esquema navegacional a nivel agrupamiento de elementos (es decir clases), y no representando los enlaces entre elementos concretos (como hacían el modelo Trellis o el de hipergrafos). Esto presenta un aumento de la legibilidad de los diagramas de navegación. Por otro lado la semántica de navegación se determina en función de todos los esquemas del modelo.

Técnicas orientadas a objetos. OOHDM utiliza con profusión las técnicas orientadas a objetos en todos los niveles. Esta utilización permite representar no solo las actividades típicas de cualquier aplicación hipermedia, sino, cualquier actividad computacional asociada a la aplicación. Como ya hemos comentado esta potencia de especificación tiene la contrapartida de tener que especificar características, en principio implícitas, a una aplicación hipermedia.

2.7 Conclusiones

En este capítulo hemos analizado nueve sistemas de representación hipermedia de muy distinta naturaleza. Todos presentan ventajas e inconvenientes, y debido a la heterogénea naturaleza de las aplicaciones hipermedia no podemos decir que unos sean mejores que otros. No olvidemos que hoy en día existe una gran cantidad de páginas HTML que se ajustan perfectamente al modelo descrito por la máquina abstracta de hipertextos, la cual puede ampliarse con los niveles descritos en el modelo Dexter. También hay aplicaciones hipermedia con una fuerte componente multimedia, en cuyo caso necesitarían aplicar un sistema de representación capaz de representar totalmente estas capacidades, como el Amsterdam. La gran cantidad de información existente en formato relacional hace de los modelos HDM y RMDM la herramienta idónea para proporcionar una interfaz simple e intuitiva de las bases de datos. Si la aplicación contiene gran cantidad de procesos computacionales, o contenidos dinámicos, Labyrinth y OOHDM son la opción más adecuada. Finalmente, modelos como el de hipergrafos o Trellis sirven para aplicaciones no tan complejas como las que necesitarían la aplicación de Labyrinth u OOHDM, pero no tan simples, como las descritas por HAM. Si abstraemos las principales características de cada sistema de representación hipermedia, y hacemos una comparación en formato tabular, obtenemos los resultados descritos en la Figura 2.57.

	a	b	c	d	e	f	g	h	i	j
Dexter	✓	×	×	×	×	×	×✓	×	×	×
Amsterdam	✓	✓	✓	×✓	×	×	×	×	×	×
HAM	×	×	×	✓	×	×	×	×✓	×	×
Hipergrafos	✓	×✓	×	✓	×	×	×	×✓	×	×
Trellis	✓	×✓	×✓	✓	×	×	×	×✓	×	×
HDM	✓	×	×	×✓	✓	×	×	×	×	×
RMM	✓	×	×	×✓	✓	×	×✓	×	×✓	×
Labyrinth	✓	✓	✓	×	×	✓	✓	✓	×	×✓
OOHDM	✓	✓	✓	×	×✓	✓	✓	×	✓	✓

a) Separación contenidos, navegación y presentación
b) Contexto
c) Sincronización
d) Formalización semántica de navegación
e) Organización relacional de los contenidos
f) Gestión explícita de eventos
g) Contenidos dinámicos
h) Gestión de usuarios y seguridad
i) Distinción enlaces entre contenidos y navegacionales
j) Técnicas orientadas a objetos

✓: Si
×: No
×✓: Parcialmente o con ligeras modificaciones

Figura 2.57 Características de los sistemas de representación hipermedia

Como podemos ver salvo *i)* no existe ninguna característica distintiva en los sistemas de representación hipermedia. *Pipe*, el modelo hipermedia propuesto en esta tesis, si que presentará una distinción entre los enlaces de contenidos y los enlaces navegacionales. Además Pipe estará preparado para tratar con contenidos dinámicamente generados, proporcionando semántica de presentación formalizada. Por supuesto, también presenta ventajas tan importantes como la separación de contenidos/navegación/presentación, y la sincronización. Por lo tanto si ampliamos la tabla anterior con el modelo Pipe, obtenemos la Figura 2.58.

	a	b	c	d	e	f	g	h	i	j
Dexter	✓	×	×	×	×	×	×✓	×	×	×
Amsterdam	✓	✓	✓	×✓	×	×	×	×	×	×
HAM	×	×	×	✓	×	×	×	×✓	×	×
Hipergrafos	✓	×✓	×	✓	×	×	×	×✓	×	×
Trellis	✓	×✓	×✓	✓	×	×	×	×✓	×	×
HDM	✓	×	×	×✓	✓	×	×	×	×	×
RMM	✓	×	×	×✓	✓	×	×✓	×	×✓	×
Labyrinth	✓	✓	✓	×	×	✓	✓	✓	×	×✓
OOHDM	✓	✓	✓	×	×✓	✓	✓	×	✓	✓
Pipe	✓	✓	✓	✓	×	×	✓	×	✓	×

a) Separación contenidos, navegación y presentación
b) Contexto
c) Sincronización
d) Formalización semántica de navegación
e) Organización relacional de los contenidos
f) Gestión explícita de eventos
g) Contenidos dinámicos
h) Gestión de usuarios y seguridad
i) Distinción enlaces entre contenidos y navegacionales
j) Técnicas orientadas a objetos

✓: Si
×: No
×✓: Parcialmente o con ligeras modificaciones

Figura 2.58 Comparación de Pipe con el resto de sistemas de representación

Si observamos la Figura 2.58 vemos como ningún modelo soporta la noción de contexto y de generación dinámica de contenidos proporcionando un formalismo explícito para representar el esquema navegacional y la semántica de navegación (*b)*, *d)* y *g)*), salvo Pipe. Además es el único, junto a OOHDM, que presenta una total separación los enlaces de contenidos y los enlaces navegacionales (*i)*). Nótese que el modelo no se preocupa de caracterizar eventos (*f)*), ni de organizar los contenidos (*e)*). Pensamos que estos dos puntos son de vital importancia a la hora de construir una aplicación hipermedia, pero que no tienen cabida en un modelo que proporcione una semántica de navegación formalizada. En esta tesis, el modelo Pipe se utilizará

para dirigir la fase de conceptualización de los modelos de proceso Plumbing y PlumbingXJ, inspirados ambos en el modelo de proceso de Fraternali.

3 Ingeniería hipermedia

3.1 Introducción

Aunque existen diversas definiciones de ingeniería del software [Pressman 98], una descripción bastante gráfica es aquella proporcionada por Booch en su libro de análisis y diseño orientados a objetos [Booch 96]. Según esta, un avión es un objeto cuya tendencia natural es caer, y sobre el cual se aplican multitud de esfuerzos para evitar que suceda. Pues bien, la experiencia nos demuestra [Pressman 93], [Pressman 01], [Sommerville 01], [Booch 96] que un proyecto software es similar a un avión. Su tendencia natural es caer, y debemos aplicar multitud de esfuerzos para evitar que esto suceda. Dichos esfuerzos se concretan en la disciplina de *Ingeniería del Software*, tanto en su vertiente de *gestión*, como en su vertiente de *desarrollo técnico*.

La gestión de proyectos software podríamos definirla como todos los esfuerzos complementarios al desarrollo técnico de la aplicación que se llevan a cabo para garantizar un desarrollo racional de la misma. El desarrollo técnico centra en caracterizar una serie de técnicas concretas de construcción de aplicaciones, prestando especial atención al *análisis y diseño*, de tal forma que no solo se garantice su corrección y completitud frente a las especificaciones, sino que además se consiga un alto nivel de mantenibilidad.

La reconciliación entre ambas facetas de la ingeniería del software viene de la mano del *modelo de proceso del software*. Podemos concebir a dicho modelo de proceso como las actividades que se llevan a cabo para construir un proyecto software. Evidentemente características deseables de estas actividades es que estuviesen bien definidas, fueran repetibles, disminuyesen el tiempo de desarrollo, y en general que presentasen todos las buenas características de las tareas un modelo de proceso [Pressman 01]. La integración de las dos facetas dentro del modelo de proceso es una cuestión tan simple como considerar actividades del modelo de proceso de gestión, y actividades de desarrollo. A pesar de estas descripciones ortogonales, el modelo de proceso, la gestión del proyecto, y las técnicas de análisis y diseño (y por extensión de desarrollo) están íntimamente relacionadas. Por ejemplo, la culminación del proceso de gestión del proyecto es el *plan del proyecto*, el cual debe incluir que modelo de proceso se está utilizando, pero es precisamente dentro de una etapa de dicho modelo de proceso cuando se crea el plan de proyecto.

Aunque en principio, los modelos de proceso son independientes de la naturaleza de la aplicación, también es reconocido que los modelos de proceso tienden a personalizarse en cada entidad de desarrollo de software, y en la medida que estas entidades se dediquen al desarrollo de aplicaciones hipermedia, este modelo tenderá a especializarse en este tipo de aplicaciones [Pressman 01]. El modelo unificado de desarrollo [Jacobson 00], donde se utiliza una notación de análisis y diseño concreta (y en consecuencia un paradigma de implementación) puede ser un buen ejemplo de este hecho. Por lo tanto parece razonable adaptar modelos de proceso genéricos para el desarrollo de aplicaciones hipermedia. Además según [Olsina 98] los modelos de proceso clásicos no se adaptan bien al desarrollo de aplicaciones hipermedia, ya que en estas aparecen características particulares que las diferencian de las aplicaciones tradicionales. Entre estas características podemos señalar una mayor importancia del papel desempañado por el cliente, ya que ahora no solo proporciona la especificación de la aplicación, sino además los contenidos y el esquema navegacional que se va a imponer sobre estos.

A pesar de este hecho, no existen demasiados modelos de proceso documentados específicos para aplicaciones hipermedia [Lowe 99b]. En particular disponemos de los propuestos por [Fraternali 98], [Ginige 97], y [Nanard 98]. Otros como Lowe [Lowe 99a], [Lowe 99b] introducen modelos de proceso de referencia, con el único fin de evaluar y mejorar los modelos de proceso hipermedia.

Este capítulo está centrado en la intersección de la ingeniería del software con el desarrollo hipermedia, área que podríamos denominar *Ingeniería del Software Hipermedia, o Ingeniería Hipermedia* para abreviar. En él estudiaremos los modelos de proceso mencionados, así como las metodologías concretas basadas en sistemas de representación hipermedia. Aunque existen multitud de metodologías de desarrollo de este tipo de aplicaciones, en esta tesis nos interesaremos exclusivamente en aquellas basadas en los sistemas de referencia hipermedia comentados en el Capítulo 2, ya que son las que presentan una aproximación más formal, y al mismo tiempo son las de mayor calado en el dominio hipermedia. Estas metodologías utilizan sistemas que mediante estructuras basadas en sistemas de representación hipermedia son capaces de generar aplicaciones, o como mínimo sirven como base para el desarrollo de dichas aplicaciones. Como excepción a la inclusión de metodologías concretas de desarrollo veremos la metodología DTC. Aunque no es específica para el dominio hipermedia (y por tanto no está basada en ningún sistema de referencia hipermedia) se incluye por varias razones. La primera es que muestra una utilización distinta de los lenguajes de marcado, tal y como lo concibe

aplicaciones como SMIL, o como la propuesta en esta tesis (esta razón la hace candidata para el siguiente capítulo de lenguajes de marcado, pero como su fundamentación teórica trasciende el uso de dichos lenguajes hemos preferido incluirla en este capítulo). La segunda es que ha sido utilizada por nuestro grupo de investigación como base para el desarrollo de aplicaciones hipermedia. La tercera es que muestra un ejemplo de tecnología concreta aplicable al modelo de proceso PlumbingXJ propuesto en el Capítulo 6.

Tampoco se revisarán sistemas ni técnicas concretas de implementación, ya que nuestro objetivo no es sustituirlos ni modificarlos, si no proporcionar un mecanismo previo que facilite su aplicación. En [Fraternali 99] y [Nielsen 95] pueden encontrarse descripciones de dichos sistemas y técnicas. También hemos obviado cualquier mención al área de Ingeniería Web [Murugesan 99] por dos razones. La primera es que es un área extensa y todavía en expansión, lo cual dificulta mucho un estudio sistemático. La segunda se debe a su interés exclusivo por las aplicaciones Web, subconjunto de las aplicaciones hipermedia, lo cual repercute en un enfoque excesivamente sesgado, a nuestro entender, hacia las tecnologías de desarrollo de este tipo de aplicaciones.

3.1 Modelos de proceso

3.1.1 El modelo de proceso Fraternali

Modelo de proceso

En su artículo [Fraternali 98], Piero Fraternali propone un modelo de proceso para aplicaciones Web (que aquí extendemos a aplicaciones hipermedia en general) surgido como intersección de los modelos de proceso clásicos con las propuestas de diseño basadas en los sistemas de representación hipermedia. La Figura 3.1 recoge este modelo de proceso.

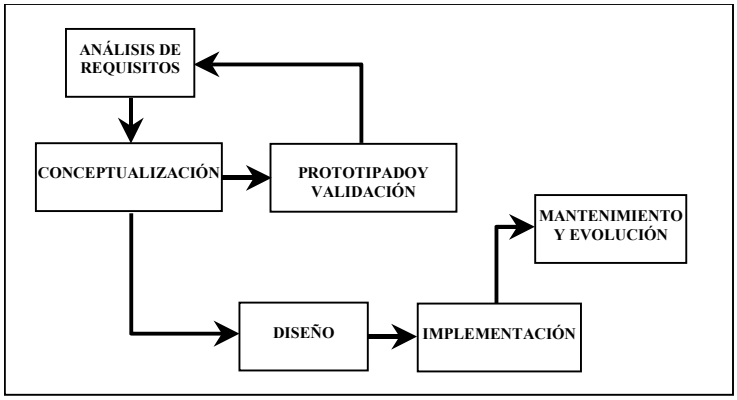

Figura 3.1 Modelo de proceso de Fraternali

En la fase de *análisis de requisitos*, se determina la misión de la aplicación identificando a los usuarios y la naturaleza de la información. Además de los requisitos del cliente y de los análisis de viabilidad, las aplicaciones que requieran adaptación al usuario [De Bra 00] deben prestar especial atención a la interacción hombre-máquina, para determinar la interacción más adecuada para cada tipo de usuario, y para cada tipo de dispositivo en el que se va a mostrar la información.

En la fase de *conceptualización*, se obtiene una representación de la aplicación a través de un conjunto de modelos abstractos que representan los componentes principales de la solución propuesta. En el contexto hipermedia, la conceptualización varia respecto a la misma actividad que en diseño clásico, ya que el interés se centra en representar los objetos, sus relaciones y el esquema navegacional que se va a imponer sobre estos, en vez de cómo serán representados en la aplicación software. En particular en esta etapa se debe caracterizar la *estructura* que describe la organización hipertextual de la información gestionada por la aplicación, en términos de las piezas que constituyen su contenidos base y sus relaciones semánticas. También debe caracterizar la *navegación* centrada en las facilidades para acceder a la información y para moverse a través del contenido de la aplicación. Por último también debe centrarse en la *presentación* que afecta a la forma en que el contenido y la navegación se presenta al usuario. Esta fase es previa a la de diseño, y es importante darse cuenta que incluso utilizando diagramas entidad/relación para caracterizar la estructura, los esquemas resultantes de la conceptualización y de la aplicación variarán normalmente.

En la fase de *prototipado y validación*, se muestran al usuario versiones simplificadas de la aplicación para obtener una realimentación previa al diseño. En el caso de aplicaciones hipermedia el prototipado adquiere una especial relevancia respecto a las aplicaciones software tradicionales. En efecto, debido a la complejidad de la interfaz de usuario de las aplicaciones hipermedia se requiere una cuidadosa evaluación por parte del cliente de la efectividad de la solución, tanto a nivel estructural como de navegación. Normalmente se construye un prototipo previo al diseño y sobre una arquitectura simplificada, como por ejemplo, un conjunto de páginas HTML que contengan ejemplos de los contenidos de la aplicación emulando el comportamiento y apariencias deseados.

En la fase de *diseño*, los esquemas conceptuales de la fase de conceptualización se transforman en representaciones de bajo nivel, cercanas a las necesidades de implementación, pero todavía dependientes del contenido real de la información base. Normalmente la visión estructural de los contenidos se traduce al formato de datos utilizado por la aplicación (relacional, o XML, por ejemplo), y el esquema navegacional y presentacional a un conjunto de primitivas de acceso y presentación sobre la información. Además en esta fase debe decidirse la arquitectura de la aplicación, la cual refleja la disposición física de los datos de la aplicación, así como la distribución espacio temporal de los procesos computacionales que se van a llevar a cabo durante la ejecución. En aplicaciones hipermedia tradicionales se optará por una arquitectura de una sola capa en la que solamente existirá un único ordenador donde almacenar los datos y ejecutar los procesos computacionales. En aplicaciones Web podemos optar por la arquitectura de dos capas, donde los clientes (navegadores) son aplicaciones responsables de presentar la información, mientras que el servidor es el responsable de almacenar los datos y de ejecutar la lógica de la aplicación. También podemos utilizar arquitecturas más avanzadas de tres o varias capas, donde separamos la lógica de la aplicación y los datos en dos o varios servidores independientes (servidores de aplicaciones y de datos). Esta última opción permite la implementación de arquitecturas avanzadas que integran el tradicional protocolo HTTP, junto a protocolos de aplicación cliente servidor distribuidos para obtener un mejor rendimiento, escalabilidad, robustez y seguridad. Otra arquitectura ortogonal a la anterior se centra en la relación entre la información base y la que recibe el usuario. De esta manera podemos tener aplicaciones *estáticas* donde la información se proporciona antes de la ejecución de la aplicación, y no cambia durante la misma, o aplicaciones *dinámicas*, donde la información se genera en tiempo de ejecución a partir de la información base.

En la fase de *implementación*, se proporcionan los contenidos reales de la aplicación por parte de expertos. Este contenido se codificará de acuerdo del formato de representación elegido. También se procederá a la traducción de los esquemas de diseño a un lenguaje determinado (HTML, Java, ActiveX, o Authorware, por ejemplo). Evidentemente esta transición será más o menos lineal en función del lenguaje de diseño y de implementación elegidos, así como su correlación.

La fase de *evolución y mantenimiento*, se produce después de la entrega, y se encarga de los cambios en los requisitos del cliente, o en la solución de defectos que hayan podido trascender a las actividades de protección que deben acompañar a todo modelo de proceso [Pressman 01]. Estas actividades pueden requerir cambios en la estructura, navegación, presentación o contenidos. La mejor manera de llevar a cabo estos cambios es volver a repetir el modelo de proceso, dirigido esta vez por mecanismos de gestión de la configuración software.

El modelo de proceso propuesto por Fraternali puede concretarse en multitud de procesos reales de desarrollo, dependiendo su aplicabilidad del contexto específico de desarrollo. Dentro de este contexto cabe citar la disponibilidad de soporte CASE, la complejidad de la aplicación, y la frecuencia del cambio. Por lo general y con aplicaciones de tamaño limitado, y con requisitos estables, tras el análisis de requisitos, y varias interacciones a través de prototipado, se puede pasar directamente a la fase de implementación. La importancia de la conceptualización y el diseño aumenta de manera directamente proporcional a la volatilidad de los requisitos y a la complejidad de la aplicación.

De la conceptualización al diseño

Aunque lo ideal sería una transición lo más directa posible de la etapa de conceptualización a la de diseño, en la última generación de aplicaciones hipermedia, que utilizan una pléyade de heterogéneas técnicas de implementación, esta transición no puede ser tan sencilla. Por ejemplo supongamos una aplicación hipermedia educativa de varias capas para su uso en la Web. Su arquitectura software va a ser la siguiente: tendremos contenidos generados dinámicamente los cuales se producen en base a diversos criterios de

adaptación al usuario. Además la comunicación entre cliente y servidores, y entre estos, va a ser mediante XML. Por último se va a proporcionar una implementación basada en componentes software. El diseño de dicha aplicación involucra conceptos de aplicaciones Web, componentes, bases de datos, XML y de diversos protocolos de comunicación. Es evidente que salvo sistemas de representación hipermedia capaces de representar actividades computacionales genéricas como OOHDM o Labyrinth (ayudados probablemente por UML), el resto de sistemas de representación son incapaces de representar esta arquitectura. De hecho la mayoría fueron diseñados antes de la existencia de dichas técnicas. Podríamos por tanto utilizar estos sistemas de representación hipermedia para la fase de conceptualización, pero es aquí donde surge el mayor inconveniente de OOHDM y Labyrinth: no cuentan con semántica de navegación por defecto. Esto nos obliga a programar de manera manual prototipos que serán desechados en última instancia, ya que estarán construidos en base a arquitecturas mucho más sencillas. Además el utilizar estos sistemas de representación en la fase de conceptualización, previa al diseño, obliga a un nivel de detalle difícilmente alcanzable en esta etapa. El problema radica en utilizar mecanismos ideados como herramientas de diseño, para ayudar a la creación del propio diseño. En última instancia esto hecho repercute en el poco uso de los sistemas de representación hipermedia en el desarrollo de aplicaciones hipermedia industriales [Barry 01].

Sería por tanto deseable disponer de sistemas de representación hipermedia que aunque no representen todos los detalles de la arquitectura de la aplicación (no olvidemos que en conceptualización no es este nuestro objetivo), sirvan para caracterizar perfectamente los contenidos de la aplicación hipermedia, sus relaciones, y su semántica navegacional, permitiendo al mismo tiempo una generación automática de la aplicación en base a dichas descripciones [Nanard 98]. Nótese que para esta última característica es indispensable que los sistemas de representación hipermedia cuenten con una semántica de navegación predeterminada. También sería deseable que fuesen capaces de modelar contenidos generados dinámicamente, característica clave en las últimas aplicaciones desarrolladas para la Web [Bodner 00], [Göschka 99]. Por último es indispensable que estos sistemas de representación pudieran representar la información proporcionada por el modelo Dexter, incluyendo en gran medida las capacidades de sincronización y de representación de contextos [Hardman 93]. Pues bien, ninguno de los sistemas de representación hipermedia citados en el Capítulo 2 cumplen todas estas características de manera simultanea [Navarro 02]. En esta tesis doctoral se formalizará el modelo Pipe, el cual cubre todas estas características junto con un alto nivel de abstracción. De esta forma se convierte en un candidato óptimo para materializar con éxito la fase de conceptualización del modelo de proceso de Fraternali.

Pipe no fija ninguna técnica concreta de implementación, dando como resultado de su fusión con el modelo de proceso propuesto por Fraternali el modelo de proceso Plumbing descrito en el Capítulo 6. Si en Plumbing comprometemos técnicas concretas de construcción de prototipos y/o aplicaciones, como Java y XML, obtendremos el modelo de proceso (que en última instancia puede concretarse en metodología) PlumbingXJ descrito también en el Capítulo 6.

3.1.2 El modelo de proceso de Ginige y Lowe

Modelo de proceso

Este modelo de proceso fue presentado por Ginige y Lowe en un tutorial en el Hypertext 97 [Ginige 97], aunque el material aquí referenciado está extraído de [Wills 98].

El modelo de proceso proporciona una clara distinción entre los procesos de diseño y autoría. Los autores reúnen los procesos de diseño en un grupo denominado *Proceso Creativo*. Esta es una etapa multidisciplinar que involucra a expertos del dominio, diseñadores, programadores, diseñadores gráficos, especialistas en interacción hombre-máquina, etc. La idea fundamental es incluir la especificación en el proceso, donde esta debe cubrir: características técnicas, contenidos básico, y características cognoscitivas. Esta especificación es la base del diseño, el cual debe centrarse en: la estructura de la información, el método de acceso a la información (esquema navegacional), apariencia, y las bases para el desarrollo del contenido. En particular este proceso se divide en cuatro etapas. En le etapa de *especificación* se determinan los requisitos que debe cumplir la aplicación a desarrollar. Tras esta fase viene el *diseño* donde se proporciona una representación independiente del código de la aplicación hipermedia. Por último dentro de este proceso viene la etapa de *evaluación* por parte del cliente. Tras esta fase puede optarse por iterar el ciclo, a partir de la etapa de diseño o proceder con el proceso mecanizado.

En el *Proceso Mecanizado* se procede al desarrollo de la aplicación final. En esta fase se suponen los requisitos de la aplicación estables y por tanto puede procederse a su producción a gran escala. Se inicia por una etapa de *diseño para producción*, donde se proporciona un diseño ligado a las características reales y definitivas de la aplicación. Tras esta viene la etapa de *desarrollo*, donde se crea la aplicación hipermedia en base al diseño de producción. Finalmente tras una etapa de *prueba* de los productos de la anterior fase se puede optar por iterar este ciclo, o par pasar a la creación de la aplicación definitiva. La Figura 3.2 recoge este modelo de proceso.

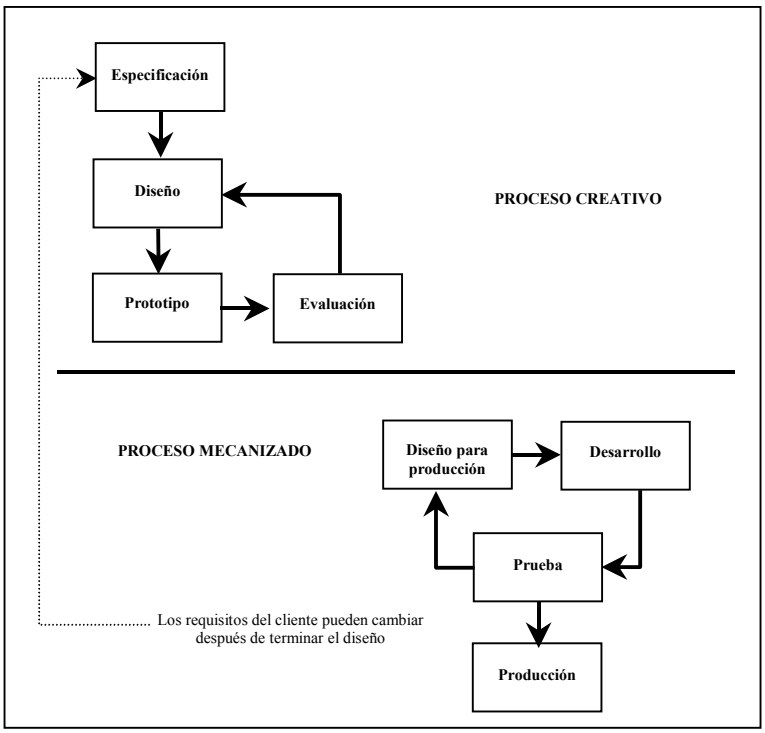

Figura 3.2 Modelo de proceso de Ginige y Lowe

Como podemos observar este modelo de proceso es bastante similar al de Fraternali, donde la conceptualización ahora se denomina diseño, y el diseño ahora se denomina diseño para producción. También hace explícita la realimentacion entre el diseño de producción y la producción. Si fusionamos los modelos de proceso de Fraternali con el de Ginige y Lowe obtenemos un tercer modelo de proceso que desde nuestro punto de vista es el más razonable para el desarrollo de aplicaciones hipermedia. La Figura 3.3 recoge este modelo de proceso mixto. Este será el modelo de proceso utilizado para introducir el modelo Pipe en la fase de conceptualización. Dicho modelo lo denominaremos Plumbing (Capítulo 6).

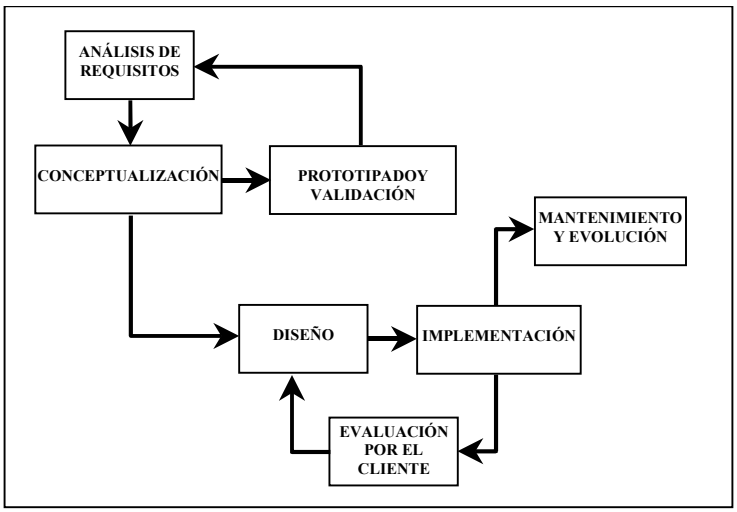

Figura 3.3 Fusión del modelo de proceso Fraternalli con el de Ginige y Lowe

3.1.3 El modelo de proceso de Nanard y Nanard

Modelo de proceso

El trabajo de Nanard y Nanard [Nanard 98] parte de una crítica a los modelos de proceso clásicos en el mundo hipermedia. En estos tras una fase de diseño abstracto en la que se identifican los componentes de contenidos, navegación y presentación de las aplicaciones hipermedia se procede a la construcción de la aplicación hipermedia y a la evaluación por parte del cliente. Es decir, el modelo de proceso lineal-secuencial [Pressman 98] aplicado al campo hipermedia. La Figura 3.4 recoge este modelo de proceso.

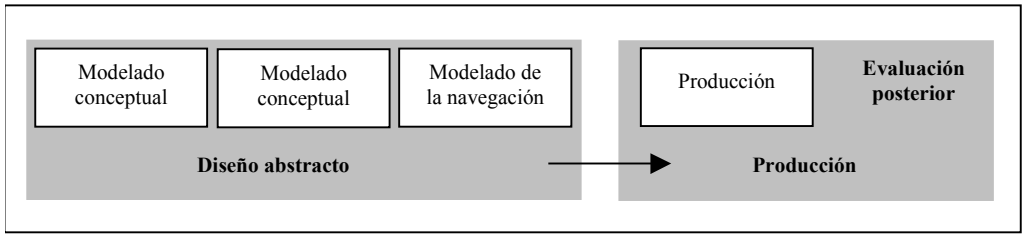

Figura 3.4 Ciclo de vida clásico

Las críticas que sufre este ciclo de vida coinciden con las críticas tradicionales al modelo lineal. Como solución, los autores proponen un modelo de proceso incremental que utilice herramientas CASE en la fase de diseño para generar de manera automática aplicaciones hipermedia a partir de especificaciones. En particular esta herramienta CASE está concebida como una interfaz entre el diseñador y el proceso de diseño, en vez de una herramienta para describir, editar y producir aplicaciones hipermedia. La Figura 3.5 describe la relación entre el diseñador y la herramienta. Aunque según esta figura realmente proponen un modelo de proceso, como veremos más adelante en último término proporcionan una metodología concreta de desarrollo de aplicaciones hipermedia. Hemos decidido incluirla aquí, en vez de en la sección siguiente, ya que los autores no conciben el modelo de proceso con independencia de la metodología [Nanard 98].

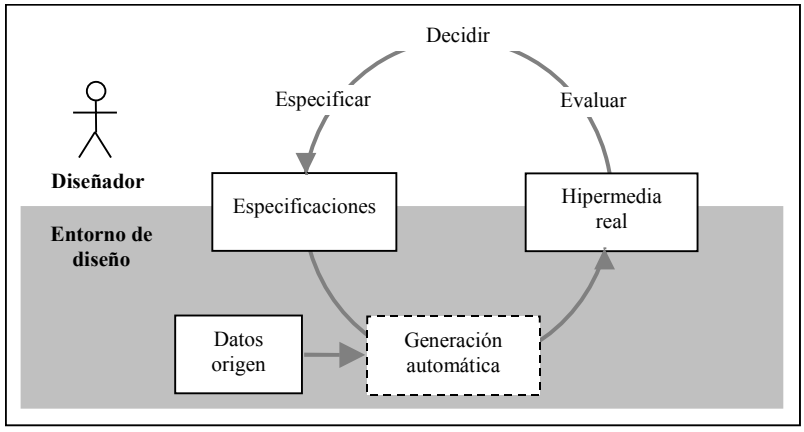

Figura 3.5 Cooperación entre el diseñador y el entorno de desarrollo

El modelo de proceso se basa en tres puntos fundamentales: construir un diseño incremental basado en evaluaciones de la aplicación real con usuarios reales, proporcionar un marco que permita especificar el diseño para permitir la generación automática con el fin de optimizar el bucle anterior, y facilitar la interacción entre el diseñador y su trabajo. Por tanto, el proceso de diseño se organiza como un ciclo.

Para aplicar el modelo anterior se necesita una herramienta de generación automática de aplicaciones. La arquitectura de producción descansa en tres componentes fundamentales: los *datos origen*, que es de donde parte el diseñador, la *hipermedia destino*, que es lo que el diseñador intenta producir, y finalmente las *especificaciones de generación*, que indican a la herramienta como transformar los datos origen en la hipermedia destino. Los datos origen representan el contenido de la aplicación, es decir, las imágenes, textos, vídeos, etc. Tienen su propia estructura con independencia de su uso en la aplicación. Esta estructura se divide en dos partes, el *modelo de datos conceptual* (una DTD XML o un diagrama entidad-relación, por ejemplo), y la *estructura de datos real* (la instancia de la DTD, o la instancia de la base de datos). La hipermedia destino es el resultado del proceso de diseño y producción. Su estructura se organiza con técnicas orientadas a objetos, y descansa

sobre los principios del modelo Dexter. La especificación de la estructura del la hipermedia destino se describe de manera genérica como clases dentro del lenguaje de especificación HLHSL (*High Level Hypermedia Specification Language*). Cualquier componente de la hipermedia destino se considera una instancia de una clase cuya estructura se especifica como una plantilla.

La terna formada por el modelado de los datos origen, reglas de generación, y especificación de la hipermedia destino define una aplicación entre los datos origen y la hipermedia destino, especificando como se puebla la hipermedia destino (Figura 3.6). Un *disparador* asociado a cada clase de la hipermedia destino define sus reglas de generación especificando que propiedades, definidas sobre el modelo de datos origen deben cumplir cualquiera de las instancias de un componente hipermedia. Con esto un analizador dispara automáticamente la instanciación de la clase siempre que la propiedad especificada se cumpla en los datos origen. Una vez que la especificación de la hipermedia destino y las reglas de generación están escritas en HLHSL, un generador transforma los datos origen en la hipermedia especificada.

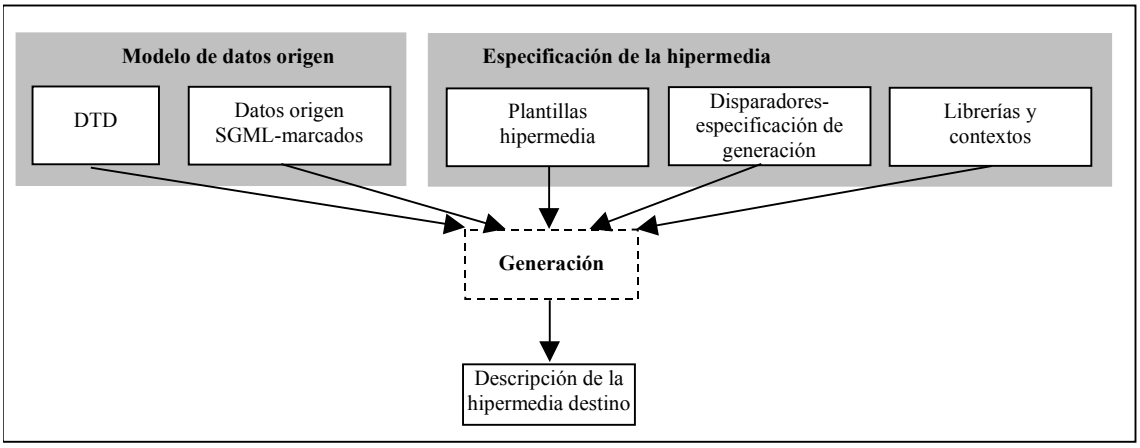

Figura 3.6 La arquitectura del proceso de generación

La arquitectura del proceso de diseño queda recogida en la Figura 3.7. La evaluación se lleva a cabo sobre las versiones reales generadas, consideradas prototipos durante un bucle de realimentación experimental. Dependiendo de la evaluación, se introducen mejoras incrementales en las especificaciones, en los datos, o en sus descripciones. Como la elaboración incremental de la aplicación está dirigida por su evaluación es posible redefinir el diseño de manera iterativa. La velocidad y el bajos coste de la generación automática permite comprobar y evaluar tantas decisiones de diseño como sean necesarias. De esta forma el diseñador puede observar y evaluar, incluso con los usuarios reales, la hipermedia producida en cualquier estado del diseño.

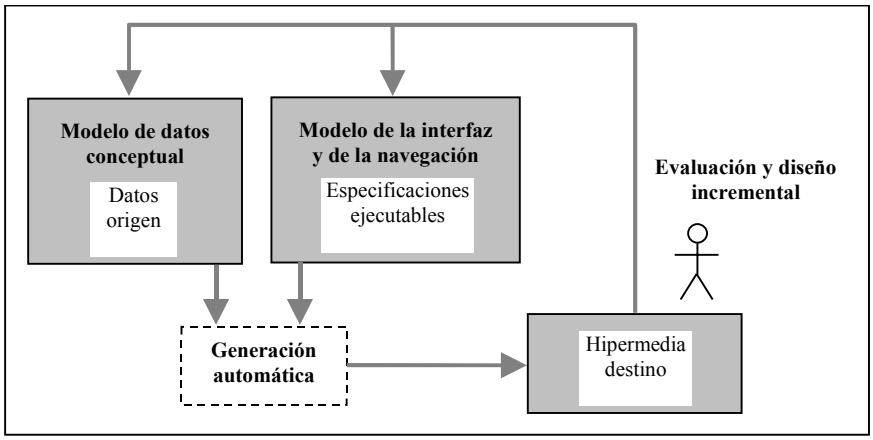

Figura 3.7 Generación a partir de las especificaciones con un ciclo de realimentación experimental

El propósito de esta arquitectura es ayudar al diseñador en el trabajo de alto nivel de abstracción refinando el diseño al actualizar las especificaciones, o los datos disponibles actualizando cada componente de la hipermedia destino. Sin embargo, elaborar especificaciones textuales suele considerarse una tarea tediosa para el diseñador. Por tanto la arquitectura del sistema de diseño se organiza con distintos fines. El primero es

soportar la manera de trabajar gráfica e incremental de los diseñadores. El segundo es proporcionar el conjunto de operaciones con el fin de pasar de especificaciones gráficas a textuales y viceversa, así como para modificar ambas descripciones. El tercero y último es ayudar a capturar como especificaciones las abstracciones usadas por el diseñador.

El propósito final es mantener el modo natural de trabajar de los diseñadores en el entorno de la arquitectura. La aproximación para elaborar especificaciones se basa en una descripción visual: el diseñador dibuja, estructura y reutiliza partes de los diagramas con ayuda del sistema, que le facilita la abstracción. La especificación no precede al diseño, si no que resulta de la abstracción de los diagramas, los cuales representan el trabajo deseado. El mecanismo formal utilizado se basa en un lenguaje de prototipado visual. Como muestra la Figura 3.8, el diseñador puede dibujar libremente cualquier objeto o estructura, por ejemplo algunas páginas de la hipermedia destino que son representantes del diseño. Estos objetos se pueden utilizar como prototipos por clonación (copia con herencia entre objetos) para construir otros nuevos. El sistema también ayuda a traducir los objetos dibujados en plantillas, que se utilizan para especificar la estructura de la aplicación, y las reglas de generación. El sistema traduce el objeto seleccionado en el modelo de una clase que se instanciará siempre que el sistema lo necesite, mejorando de esta forma la consistencia global.

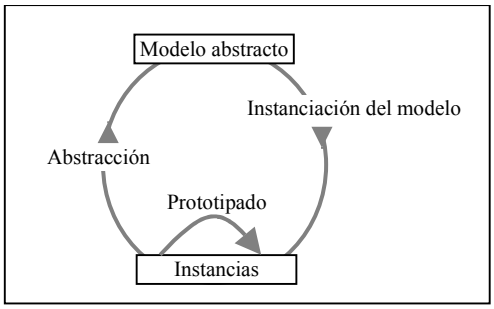

Figura 3.8 Modelo de interacción para elaborar las especificaciones

El sistema de diseño también proporciona al diseñador las operaciones necesarias para ejecutar el ciclo de abstracción/especialización incremental, y para editar o reutilizar las instancias gráficas, así como sus especificaciones textuales. La especificación de la hipermedia destino es un conjunto de plantillas que el diseñador ha abstraído de los objetos dibujados para hacer surgir las ideas. Aunque las plantillas se almacenan como texto, para mejorar la claridad, el entorno ayuda gráficamente a especificar gran parte de ellos. El mecanismo de plantillas ofrece una aproximación modular para encapsular toda la información necesaria para especificar y manejar cada abstracción del diseño. Una Plantilla captura la especificación de las estructuras reutilizables y los componentes durante el diseño, así como las reglas para poblar la hipermedia destino.

3.1.4 Conclusiones

Como hemos podido comprobar los modelos de proceso existentes para aplicaciones hipermedia no difieren en exceso entre si, ni tampoco son muy diferentes de los modelos evolutivos clásicos. De los tres modelos de proceso analizados, sin duda alguna, los de mayor abstracción son los de Fraternali y el de Ginige-Lowe. El de Nanard-Nanard, aunque parte de una descripción genérica e independiente de cualquier técnica de implementación (Figura 3.5) termina ligándose irreversiblemente a técnicas concretas de implementación (aunque un comportamiento similar se produce por ejemplo en el Modelo Unificado de Desarrollo). Por otro lado, todos los modelos de proceso para aplicaciones hipermedia prestan especial atención a la fase de conceptualización, y a los prototipos generados a partir de esta fase, como vehículo de comunicación entre clientes y desarrolladores. Fraternali y Ginige-Lowe independizan estos prototipos de la aplicación final, mientras que Nanard-Nanard los ven como iteraciones sucesivas que nos aproximan a la aplicación destino.

Ambas aproximaciones presentan sus ventajas y sus inconvenientes. La ventaja principal de independizar los prototipos de las versiones finales es que permiten acometer rápidamente la fase de conceptualización prestando especial interés a las características fundamentales de las aplicaciones hipermedia: contenidos, relaciones, y esquema navegacional. El mayor inconveniente radica en ligar el diseño de la aplicación a un sistema de representación hipermedia concreto utilizado en la conceptualización, limitando de esta forma el diseño de la aplicación. Si se desea descartar el prototipo inicial, tal y como indican las técnicas estándares de ingeniería del software [Pressman 01], se cuenta con el inconveniente de no poder reutilizar los prototipos en

la construcción de la aplicación. La ventaja principal de utilizar un lenguaje como HLHSL para especificar y generar automáticamente los prototipos, hasta llegar a la aplicación final, es el ahorro de esfuerzo en la programación de la aplicación, y la valiosa realimentación proporcionada por los clientes. El mayor inconveniente es ligar nuestro diseño a la potencia expresiva/generativa de nuestras especificaciones.

En esta tesis se propondrá un modelo de proceso que intente beneficiarse de ambas aproximaciones. Con este fin se propondrá un modelo hipermedia genérico para cubrir la fase de conceptualización del modelo de proceso de Fraternali con influencias de Ginige y Lowe. Dicho modelo, llamado Pipe, será presentado en el Capítulo 5. En la fase de prototipado y validación los desarrolladores deben elegir el formalismo concreto más adecuado a sus necesidades de tal forma que permita representar las estructuras Pipe generadas en la anterior fase. A partir de estas estructuras, y gracias a la semántica de navegación predeterminada por el modelo hipermedia, será posible generar automáticamente prototipos en la fase de conceptualización.

Al modelo de proceso resultante de la inserción de Pipe en el modelo de Fraternali/Ginige-Lowe lo denominaremos Plumbing y será presentado en el Capítulo 6. Si no se desea utilizar técnicas de representación ad hoc del modelo Pipe, se puede utilizar XML y Java para cubrir la fase de prototipado y validación. En principio no se desea comprometer ninguna técnica de diseño y/o implementación. De esta manera se pueden desechar totalmente los prototipos, tal y como se establece en [Presman 01], o reutilizarlos en parte. Este último supuesto se basa en que la información del modelo Pipe estará codificada en XML, existiendo multitud de herramientas disponibles para reutilizar información marcada según este estándar. El modelo de proceso que utiliza estas técnicas de prototipado se denomina PlumbingXJ y será presentado en el Capítulo 6.

Nótese que lo seguimos denominando modelo de proceso, y no metodología, ya que al igual que con multitud modelos de procesos, el utilizar una técnica concreta de especificación de requisitos, o de diseño, no compromete la naturaleza abstracta de modelos de procesos genéricos [Pressman 01], [Sommerville 01]. Por otro lado, si se aumentase la potencia del generador de prototipos podría reutilizarse en su totalidad la información generada en anteriores fases, utilizando el modelo Pipe como herramienta de conceptualización y de diseño, y obteniendo de esta forma una metodología concreta. En ese caso estaríamos en aproximaciones similares a las de Nanard-Nanard o a la de SMILE. Voluntariamente decidimos no abordar este punto ahora y dejarlo como trabajo futuro.

3.2 Metodologías basadas en sistemas de referencia hipermedia

3.2.1 Dexter, HAM, hipergrafos y HDM

En este grupo se enclavan los sistemas de representación hipermedia que no tienen ni metodología de desarrollo asociada, ni un entorno de diseño-desarrollo asociado.

El modelo *Dexter* es un modelo de referencia para caracterizar sistemas hipermedia, y por esta razón no incluye ni metodología de construcción de aplicaciones ni metodología de desarrollo. Lo único que incluye es un conjunto de funciones de las capas de almacenamiento y de ejecución que deberían incluir cualquier sistema hipermedia. Esta es la única indicación de cómo desarrollar una aplicación hipermedia.

HAM es un servidor basado en transacciones para un sistema de almacenamiento hipermedia y tampoco incluye metodología ni entorno, ya que su única finalidad es la de describir el nivel de almacenamiento de una aplicación hipermedia.

Tampoco el modelo de *hipergrafos* presenta metodología ni entorno de desarrollo. Aunque pudiera parecer extraño que un modelo hipermedia que estructura el esquema navegacional con un formalismo bien definido, y en principio relacionado con una posible semántica de navegación por defecto no tenga entorno de desarrollo, la razón es bien sencilla. Como ya comentamos en el Capítulo 2 la semántica de navegación del modelo de hipergrafos descansa en las primitivas de acceso del modelo, y no en la estructura de hipergrafo que el modelo impone sobre los hipertextos. Dicho de otra manera, el modelo de hipergrafos no cuenta con semántica de navegación por defecto, de ahí la dificultad de construir un entorno de desarrollo basado en el modelo.

Aunque en su artículo original [Garzotto 93] los autores sugerían el comienzo de la construcción de un entorno de desarrollo original, parece ser que al final HDM solamente se utiliza como una notación de diseño sin soporte CASE [Barry 01]. En principio esto parece un serio inconveniente, ya que gran parte de la potencia de HDM radicaba en la semántica de presentación por defecto, y en la traducción automática de los enlaces abstractos en concretos. Si dicho proceso debe hacerse manualmente por parte del diseñador/programador, esta puede ser la razón de la falta de aceptación de este modelo para el diseño de aplicaciones hipermedia en la industria [Barry 01].

3.2.2 Amsterdam y Trellis

En este grupo se enclavan los sistemas de representación hipermedia (modelo en este caso) que no definen ninguna metodología de construcción, pero que si cuentan con un entorno de diseño-desarrollo.

El modelo *Amsterdam* no incluye una metodología de diseño, pero si varios entornos de desarrollo basados en el propio modelo [Hardman 93], [Hardman 99]. En particular dispone al menos de los entornos de desarrollo y reproducción CMIFed [van Rossum 93], el editor y reproductor GRiNS [Oratrix 99] y el reproductor G2 de RealNetworks [RealNetworks 99]. Estas dos últimas herramientas son similares a la primera, siendo su mayor diferencia el poder editar y/o reproducir representaciones del modelo Amsterdam según el lenguaje de marcas SMIL. CMIFEd es un entorno de edición y reproducción para documentos hipermedia basado en el modelo Amsterdam. Utiliza una jerarquía estructural para representar el documento, basada en la noción de componente compuesto [Hardman 93]. Además el entorno cuenta con dos vistas más para el proceso de autoría, la vista jerárquica y la vista de canales. La primera muestra el documento como una colección de componentes estructurados que se reproducen en serie o en paralelo. La vista de canales muestra los componentes atómicos asignados a canales abstractos. Además, la información acerca de la sincronización en la aplicación hipermedia se muestra como una línea de presentación temporal. Un documento, una vez creado, puede ser reproducido el por el reproductor del entorno, el cual asigna los canales abstractos a dispositivos reales basados en las preferencias del usuario, y las capacidades del sistema.

El propósito de la vista jerárquica es mostrar y manipular la estructura de una presentación hipermedia. La estructura jerárquica de la presentación se representa en la vista jerárquica como una estructura de bloques anidados [Hardman 93]. Los rectángulos que encierran a los cuadrados representan la estructuración jerárquica del documento. El rectángulo exterior es la raíz del árbol. Los componentes que se encuentran al lado de otros se empiezan a reproducir en paralelo (al menos que existan restricciones explícitas mediante arcos de sincronización), mientras que los componentes que se encuentran más altos en el diagrama se activan antes que los que se encuentran más abajo. De esta forma, una presentación se crea definiendo la estructura de la presentación, y asignando componentes atómicos a dicha estructura. La información temporal se deduce de la estructura jerárquica y de la duración de los componentes dentro de la estructura (se pueden añadir restricciones temporales de grano fino en la vista de canales). Cuando un componente no tiene duración, como un componente temporal, se presenta al mismo tiempo que su padre. El autor puede navegar por la estructura jerárquica acercándose a las estructuras anidadas. Dentro de la vista jerárquica, el autor puede seleccionar cualquier componente atómico o compuesto e invocar un editor para modificar los datos, sea cual fuere la naturaleza de estos.

La vista de canales [Hardman 93] muestra una transformación de la vista jerárquica en términos de canales abstractos. Esta vista se presenta como una línea temporal, donde la disposición de cada componente se determina automáticamente en base a la información de este. El uso de arcos de sincronización permite especificar restricciones temporales de grano fino, entre y en medio de grupos de componentes. Los componentes atómicos que forman la presentación se muestran en su propio canal junto a su duración y restricciones temporales. Si el autor cambia la temporización en alguna parte de la presentación utilizando alguna de las vistas, la vista de canales se actualiza automáticamente para reflejar estos cambios.

El reproductor interpreta un documento hipermedia sobre el hardware disponible. También permite seleccionar los canales que deben ser reproducidos, permitiendo por ejemplo elegir un lenguaje concreto. Está muy relacionado con la vista jerárquica y de canales, permitiendo reproducir una selección de la primera (componente atómico o compuesto) o de la segunda (componente compuesto). Esto facilita la visión de una pequeña selección de la presentación sin tener que reproducirla en su totalidad. Cuando el sistema dispone de tiempo suficiente busca y carga los datos que se necesitarán en la siguiente parte de la presentación.

El modelo *Trellis* cuenta con el entorno de desarrollo αTrellis. Este entorno se concibe como prototipo de un sistema de autoría y visionado de aplicaciones hipermedia. Presenta un entorno de visionado que permite ver varios elementos al mismo tiempo, al igual que la noción de canal Amsterdam. El sistema está diseñado como una plataforma experimental, por lo que presta más atención a implementar la representación del documento a través de la red de Petri, junto a su semántica de presentación, obviando esquemas navegacionales más avanzados.

αTrellis permite la construcción y visionado de la red de Petri que representa al hipertexto. La pantalla del entorno está dividida en dos partes [Stotts 89]. En el lado derecho e encuentra el editor y simulador de la red de Petri. En el lado izquierdo se encuentra el navegador de hipertexto, subdivido en cuatro ventanas, cada una de las cuales contiene un panel de texto y un panel de botones. Utilizando el editor, el autor construye una estructura de red de Petri, y utilizando cadenas de etiquetas, especifica la asignación de lugares a elementos de texto visionables, y de transiciones a botones. La etiqueta en un lugar es el nombre de un archivo que contiene el texto asociado. La etiqueta en una transición es el nombre mostrado en un panel de botones cuando esa transición está activada y puede dispararse. Durante la navegación, cuando una marca reside en un lugar de la red de Petri, el texto en el archivo asociado con ese lugar se muestra en una de los cuatro paneles de texto. El nombre del lugar aparece en la parte superior izquierda de esa ventana. Cualquier transición activa en el postconjunto del lugar tiene sus nombres de botones mostrados en el menú de botones a la izquierda del texto. El nombre de los botones no aparece en ninguna ventana del navegador hasta que las transiciones asociadas están activadas.

Tanto el autor como el lector pueden utilizar la representación visual de la red de Petri para obtener una imagen gráfica de su posición en el documento. Solo un autor puede alterar la estructura del documento, o colocar directamente marcas en la red. La lectura del hipertexto se lleva a cabo ejecutando la red de Petri desde su marcado inicial. La ejecución se controla por la selección de los botones mostrados en la ventana del navegador, o disparando directamente las transiciones desde el editor de la red.

Como podemos comprobar mediante las descripciones, los entornos asociados al modelo Amsterdam son mucho más potentes y evolucionados que los del modelo Trellis. La razón es doble. En primer lugar el sistema del modelo Trellis es varios años anterior al del modelo Amsterdarm. En segundo lugar el modelo Amsterdam cuenta con una potencia expresiva bastante mayor que la del modelo Trellis, tal y como pudimos comprobar en el ejemplo del Capítulo 2. Lo más importante es darse cuenta de que gracias a una semántica de navegación relativamente sencilla en ambos modelos, es posible construir entornos de prototipado/desarrollo a partir de las descripciones proporcionadas por los modelos

3.2.3 La metodología de gestión de relaciones (RMM)

El modelo de datos de gestión de relaciones, RMDM, se encuentra incluido como parte de la metodología de gestión de relaciones, RMM (*Relationship Management Methodology*) [Isakowitz 95], la cual se muestra en la Figura 3.9 en el contexto del ciclo de vida del software. Al igual que con Labyrinth y OOHDM estas metodologías rozan su caracterización como modelo de proceso, pero en última instancia son metodologías ya que utilizan con profusión técnicas concretas de desarrollo e implementación. Aunque esta caracterización pudiera ser discutible, lo indiscutible es que los modelos de proceso surgen como generalizaciones de uso de metodologías concretas, lo cual les hace menos abstractos que los propuestos por Fraternali o por Ginige-Lowe.

RMM se centra en las fases diseño, desarrollo, y construcción, aunque prestando especial atención al diseño de los mecanismos de acceso (dentro de la zona gris de la Figura 3.9). Las etiquetas en las flechas muestran los productos generados durante el uso de la metodología. Los bucles de realimentación de las fases de diseño aparecen en línea discontinua. Aunque también hay bucles de realimentación entre el resto de fases, no están explícitas en la figura. El modelo de datos RMDM proporciona un importante vehículo para el diseño de la aplicación hipermedia, pero también existen otros elementos no representables mediante este modelo de datos. En particular, el modelo es incapaz de representar algo tan sencillo como un ancla dentro de un contenido.

Veamos a continuación las etapas de la metodología.

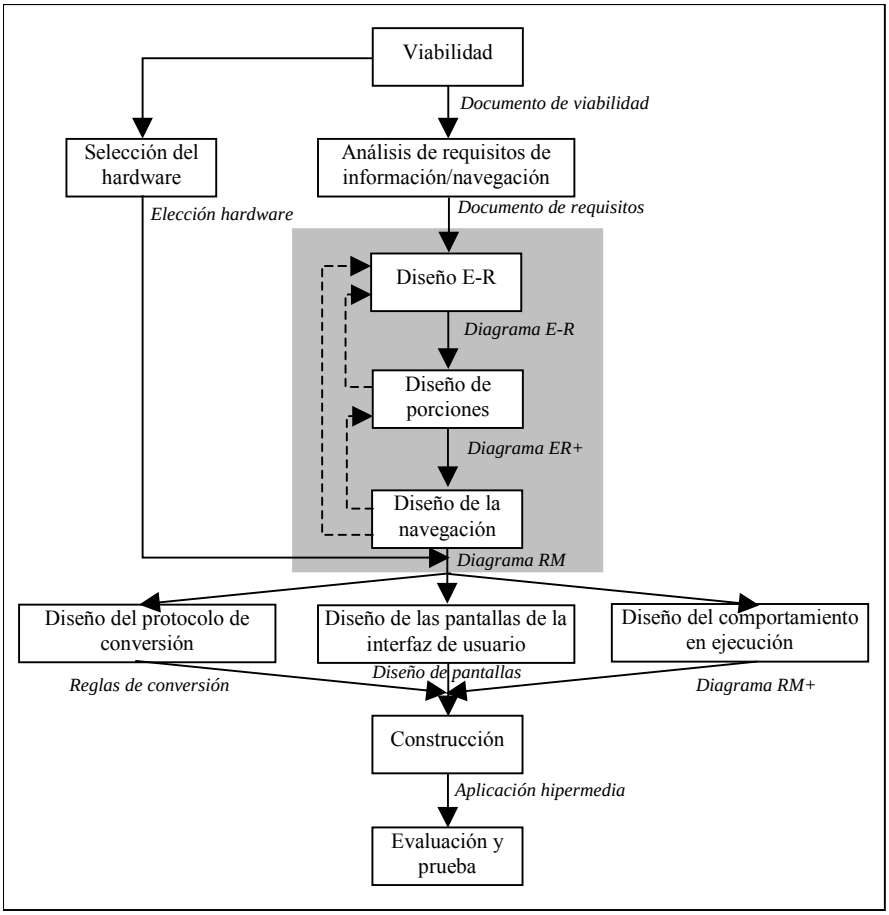

Figura 3.9 La metodología de diseño RMM en el contexto del ciclo de vida del software

Diseño E-R

El primer paso del diseño es representar el dominio de información de la aplicación mediante un diagrama Entidad-Relación. Este estado del proceso de diseño representa un estudio de las entidades y relaciones relevantes en el dominio de información. Dichas entidades y relaciones forman la base de la aplicación hipermedia, y muchas de ellas aparecerán en la aplicación hipermedia como una red de nodos y enlaces. Por supuesto también podría darse el caso de partir de un diseño relacional previo. El objetivo en el diseño de una aplicación hipermedia es hacer explícitos los enlaces entre objetos (que evidentemente se concretarán en entidades), ya que estos son el vehículo principal para que el usuario pueda acceder a la información. Según los autores un análisis del dominio utilizando la aproximación E-R ayuda a identificar las relaciones importantes sobre las cuales puede basarse la navegación. Del mismo modo, si se necesita un enlace entre entidades, es necesaria la existencia de la relación en el diagrama E-R. Esta aproximación presenta el inconveniente de su dificultad de aplicación a dominios complejos pero poco estructurados como en [Balasubramanian 97], o puede llevar a diseños "aberrantes" desde el punto de vista E-R, como en el ejemplo de la Web del profesor, recogido en la Figura 2.30 del Capítulo 2, donde a falta de una estructuración E-R natural se ha decido imponer la estructuración necesaria para soportar el esquema navegacional.

Diseño de porciones

Este paso exclusivo para aplicaciones hipermedia determina que información de las entidades elegidas se presentara al usuario, y como podrá acceder este a la misma (es decir, es algo así como una proyección del álgebra relacional [Elmasri 97]). Implica dividir la entidad en porciones significativas, y organizar estas en la red de hipertexto. En su vertiente más sencilla, toda la información de una entidad puede mostrarse en una ventana. De manera alternativa, la información puede dividirse en unidades significativas que pueden presentarse de manera interrelacionadas. Por ejemplo, de esta forma se puede dividir la información de un profesor en información general, biografía e investigación. La organización de entidades en porciones se

denomina *fase de diseño de porciones*, cuyo resultado es un diagrama de porciones. Todos los diagramas de porciones contienen una porción cabeza, que es la que se muestra al acceder a dicha entidad (algo así como la vista por defecto en HDM). El diagrama de entidades también modela la navegación entre porciones a través de enlaces uni y bidireccionales (nótese que puede hacerse así porque no hay anclas dentro de los contenidos). Estos enlaces que representan conexiones entre porciones se denominan enlaces estructurales, de manera consistente con la notación HDM. Los enlaces estructurales se diferencian de los enlaces de asociación (relaciones) en que los primeros conectan piezas dentro de la misma instancia de entidad, mientras que los segundos conectan diferentes instancias de entidad, pertenecientes por lo general a distintas clases de entidades.

Desde un punto de vista navegacional, hay una importante razón para distinguir entre estas dos conexiones. Cuando un usuario atraviesa un enlace de asociación, el contexto de información cambia, por ejemplo de un profesor a un curso. Sin embargo, cuando se navega un enlace estructural el contexto de información se mantiene dentro de la misma entidad. Para diferenciar ambos enlaces los enlaces estructurales se dibujan con líneas sólidas, mientras que los de asociación se dibujan con líneas discontinuas. La salida de esta fase es un diagrama E-R enriquecido, denotado por E-R+, el cual se obtiene rellenando cada entidad del diagrama E-R con su diagrama de porciones.

Diseño navegacional

En este paso se diseñan los caminos que permitirán la navegación por el hipertexto. Cada relación de asociación del diagrama E-R+ se analiza, y si se desea hacerla accesible para la navegación, se reemplaza por una o más estructuras de acceso RMDM. Como la metodología RMM se utiliza en dominios donde la información se actualiza con relativa frecuencia, todos los accesos navegacionales se especifican en términos genéricos. Esto quiere decir que no hay enlaces *hard coded* entre instancias de entidades, sino que los enlaces se especifican mediante propiedades de entidades y relaciones. Esto es posible gracias a los índices condicionales, tours guiados condicionales, y tours guiados indexados condicionales. De esta forma si queremos indicar que queremos que aparezca la relación entre un Profesor y las Asignaturas que imparte, no tenemos más que poner un tour guiado indexado condicional en lugar de la relación imparte, incluyendo la condición imparte(profesor, asignatura), para que solo aparezcan las asignaturas impartidas por ese profesor. Por último solo falta incluir las estructuras de acceso a las porciones por defecto (o a otras) de las entidades. Al final de esta fase el diagrama E-R+ se ha convertido en un diagrama RMDM, que describe todas las estructuras de acceso que serán implementadas en el sistema.

Interfaz de usuario y construcción

En último término hay que traducir las primitivas de acceso RMDM a primitivas de acceso del lenguaje o sistema que vaya a implementar el diseño. En el paso de *diseño del protocolo de conversión*, se utilizan un conjunto de reglas de conversión para transformar cada elemento del diagrama RMDM en un objeto de la plataforma destino. Aunque se propone un paso manual, no sería difícil automatizarlo. En el paso de *diseño de la interfaz de usuario* se diseña las pantallas de presentación para cada objeto que aparece en el diagrama RMDM. Esto incluye la disposición de botones, la apariencia de nodos e índices, y la colocación de ayudas a la navegación. En la fase de *diseño del comportamiento en ejecución* se determina si va a haber historial de selección de enlaces, vuelta atrás, y mecanismos navegacionales. También se decide si el contenido de los nodos y de los enlaces se generará antes de la ejecución o dinámicamente en tiempo de ejecución.

Finalmente en la *etapa de construcción y prueba* se llevan a cabo las actividades tradicionales de ingeniería del software, prestando especial atención a la prueba de los caminos navegacionales.

3.2.4 Ariadne

Ariadne [Díaz 99], [Montero 00] es una metodología de diseño basada en el modelo Labyrinth. La idea fundamental es representar las entidades del modelo hipermedia (entidades de bajo nivel) mediante diagramas de diseño (entidades de alto nivel). La metodología proporciona un proceso de desarrollo sistemático basado en tres fases. La primera fase, el *Diseño Conceptual*, trata el diseño desde un punto de vista abstracto, para a continuación realizar un *Diseño Detallado* donde se especifica con más detalle los elementos definidos en la fase anterior. La fase de *Evaluación* se puede ir realizando en paralelo, lo cual permite un desarrollo iterativo con

realimentación. Además para los productos generados en cada fase existe un conjunto de reglas de validación y verificación que aseguran su integridad y completitud. La Figura 3.10 representa el proceso.

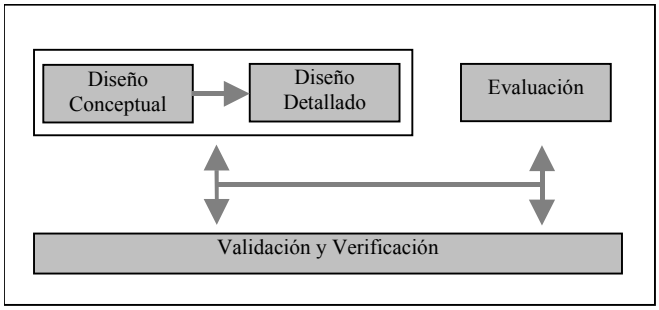

Figura 3.10 Fases de la metodología Ariadne

Asimismo, la metodología cuenta con una herramienta software en desarrollo, AriadneTool, que permite la automatización del método de diseño Ariadne

Diseño Conceptual

El objetivo de esta fase es especificar la estructura y funcionalidades del sistema desde un punto de vista lógico. Veamos a continuación los productos generados en esta fase.

El *Diagrama Estructural*, es una representación de la estructura lógica del hiperdocumento. Su misión es modelar la estructura utilizando entidades abstractas (nodos y nodos compuestos), así como las relaciones entre ellos (agregación y generalización). La estructura de hiperdocumento se especifica mediante una estructura de hipergrafo donde el origen de los hiperarcos son los nodos compuestos, el destino son sus componentes (nodos simples o compuestos, y elementos externos), y cada hiperarco es un enlace un a muchos o uno a uno cuyo tipo es agregación o generalización en función de la relación que represente.

El *Diagrama de Navegación* define la estructura de navegación del sistema presentando los diversos caminos existentes, a través de enlaces, además de incluir herramientas que faciliten el acceso directo a determinados nodos. Se construye tomando como punto de partida el diagrama estructural y definiendo entre ellos los enlaces referenciales. También se pueden definir enlaces virtuales cuyo origen o destino se define declarativamente, enriqueciendo su definición a través de eventos. Las herramientas de navegación son nodos que pueden corresponder a nodos de agregación del diagrama estructural, o a nodos simples utilizados como estructuras de acceso.

La *Especificación de Funciones* proporciona una especificación modular de los servicios no navegacionales. Se clasifican en cuatro grupos que agrupan las operaciones típicas sobre hipertextos: personalización, edición, comunicación entre usuarios y otras. La especificación de funciones sigue una aproximación descendente, ya que cada función se descompone en una serie de funciones, hasta el nivel más bajo de definición formado por eventos Labyrinth.

En el *Catálogo de Eventos* se incluyen las especificaciones de las funciones de bajo nivel. Recoge los eventos que se han definido en las especificaciones de funciones y que se han asociado a nodos o contenidos. Los eventos se pueden utilizar por ejemplo para modelar elementos creados en tiempo de ejecución, enlaces condicionales, o cualquier tipo de comportamiento interactivo.

El *Catálogo de Atributos* es un repositorio que contiene la definición de los atributos de los diagramas internos y su valor por defecto. En esta lista se incluyen los atributos obligatorios del modelo Labyrinth (Etiqueta, Autor, TipoEnlace, Dirección), y todos aquellos relevantes para definir la semántica del hiperdocumento.

En el *Diagrama Interno de Nodos*, se especifica la localización de los tipos de los contenidos, y las asignaciones de anclas, atributos y eventos a cada nodo. En esta etapa no deberían definirse posiciones específicas, pero es posible y conveniente especificar lógicamente la apariencia de un nodo y que elementos contienen un ancla origen o destino. Este diagrama está formado por dos subdiagramas: el *Diagrama Espacial*, espacio bidimensional que representa el área de visualización, y el de *Tiempo* que representa como evoluciona el nodo a lo largo de un intervalo de tiempo.

El *Diagrama Interno de Contenidos* especifica como se organizan los contenidos, además de su asignación de atributos, eventos y anclas. Al igual que los nodos, los contenidos pueden organizarse a través de relaciones de agregación y generalización.

El *Diagrama de Usuarios* identifica los tipos de usuarios que van a utilizar el sistema en forma de jerarquía. No se trata de usuarios individuales, si no de roles que representan responsabilidades o tareas de trabajo. Este diagrama, bien en forma de árbol o de red, sirve de base para la política de seguridad definida en la Tabla de Acceso, y además hace posible definir presentaciones dependientes del usuario, y personalizaciones del hiperdocumento.

Finalmente, la *Tabla de Accesos*, permite definir las políticas de seguridad del sistema, es decir, el tipo de operaciones que los usuarios van a poder realizar sobre el hiperdocumento. Para ello es necesario definir para cada papel una categoría de manipulación en cada nodo y contenido que indica qué puede hacer ese usuario en ese contexto.

Como comentábamos al principio de este punto, estas entidades tienen como fin modelar las entidades del modelo Labyrinth. La relación entre ambas puede verse en la Tabla 3.1. Dicha tabla también incluye las vistas de diseño involucradas en cada entidad de la metodología.

Entidad alto nivel	**Entidad Labyrinth**	**Vistas diseño**
Diagrama estructural	N, L, al	Datos
Diagrama de navegación	N, L, al	Navegación
Especificaciones de funciones	E	Funciones, Control
Catálogo de eventos	E	Funciones, control, navegación, presentación
Catálogo de atributos	B	Datos, navegación, presentación
Diagramas internos de nodos	N, C, el, al, lo	Datos, funciones, control, navegación, presentación
Diagramas internos de contenidos	C, el, al, lo	Datos, funciones, control, navegación, presentación
Diagrama de usuarios	U, al	Funciones, control, navegación , presentación
Tabla de accesos	N, C, ac	Datos, funciones, control, navegación, presentación

Tabla 3.1 Relación entre la metodología y el modelo

Diseño detallado

En esta fase se produce el paso de las entidades y funciones especificadas en la fase del Diseño Conceptual, que se formalizaron a un alto nivel de abstracción, a elementos más concretos del sistema que se está modelando. Las actividades que se pueden llevar a cabo son: identificar nodos o estructuras, definidas en los diagramas anteriores, para crear tantas copias como se deseen, especificar completamente las estructuras de navegación del diagrama de navegación y las funciones definidas en la especificación de funciones, detallar de una manera más específica la naturaleza de los nodos y contenidos a partir de los diagramas internos de nodos y contenidos para que se puedan crear prototipos del sistema, o identificar tipos y grupos de usuarios a partir del diagrama de usuarios para asignarles usuarios específicos.

Evaluación

Debido a la naturaleza interactiva de la hipermedia es necesario el desarrollo de prototipos durante las fases anteriores para poder realizar una evaluación que nos permita obtener información sobre la usabilidad del sistema, y mejorar las características y funcionalidades de su interfaz. Esta fase permite llevar a cabo un diseño iterativo

3.2.5 El modelo de diseño hipermedia orientado a objetos (OOHDM)

Este modelo de diseño [Schwabe 95b] surge como método de aplicación de OOHDM, y a diferencia de la mayoría de sistemas de representación hipermedia, es una metodología de diseño, en lugar de un modelo. En OOHDM, la aplicación hipermedia se construye en un proceso de cuatro pasos, basado en un modelo de proceso incremental o de prototipos. Cada paso se centra en una característica de diseño, construyéndose un modelo orientado a objetos. La Figura 3.11 muestra las fases de la metodología OOHDM.

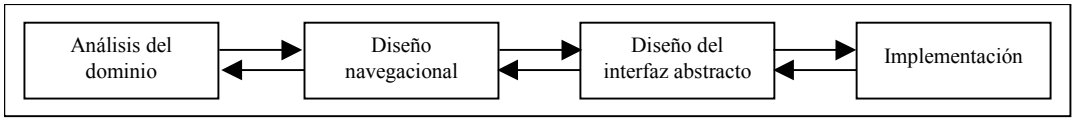

Figura 3.11 Fases de la metodología OOHDM

Análisis del dominio

El objetivo principal de esta fase es modelar la semántica del dominio de la aplicación. Para esto se construye un modelo conceptual de la aplicación utilizando principios de modelado orientados a objetos aumentados con algunas primitivas. Los productos que se obtienen en esta fase son: clases, subsistemas, relaciones, atributos y perspectivas. Las clases conceptuales se pueden construir utilizando jerarquías de agregación y de generalización/especialización. En esta fase se obvia cualquier información sobre los tipos de usuarios y sus tareas, preocupándonos solo por la semántica del dominio de aplicación. El resultado final es un esquema conceptual, formado por subsistemas, clases y relaciones.

Diseño Navegacional

El objetivo de esta fase es determinar el perfil de usuario prestando especial atención al aspecto cognoscitivo. Con este fin se describe la estructura navegacional de la aplicación hipermedia en términos de contextos navegacionales, los cuales se inducen de clases navegacionales tales como nodos, enlaces, índices y tours guiados. Los contextos navegacionales y las clases determinan los tipos de usuarios potenciales. Los nodos en OOHDM representan las ventanas lógicas de las clases conceptuales definidas durante el análisis del dominio. Es posible construir diversos modelos navegacionales para el mismo esquema conceptual con el fin de proporcionar distintas vistas del mismo dominio.

Los enlaces se derivan de relaciones conceptuales definidas en el primer paso. Al definir el esquema navegacional en términos de nodos y enlaces, se pueden modelar movimientos en el espacio navegacional (es decir, el subconjunto de nodos con los que los usuarios pueden interactuar en un momento dado) con independencia del modelo conceptual. El modelo navegacional puede evolucionar independientemente del modelo conceptual para facilitar el mantenimiento. Nótese que en último término en esta fase se produce una relación entre los objetos conceptuales y los navegacionales. En términos OOHDM los objetos producidos son los nodos, enlaces, estructuras de acceso, contextos navegacionales y transformaciones navegacionales.

Diseño de la interfaz abstracta

El modelo de interfaz abstracta se construye definiendo objetos perceptibles (cuadros, ciudades, etc.) en términos de interfaces de clases. Las interfaces de clases se definen como agregaciones de clases primitivas (como campos de texto y botones) y recursivamente sobre clases de interfaces. Los objetos de interfaz representan a los objetos navegacionales, proporcionándoles una apariencia perceptible. El comportamiento de la interfaz se declara especificando como manejar los eventos externos y generados por el usuario, y como se lleva a cabo la comunicación entre los objetos navegacionales y de interfaz. En términos OOHDM los objetos producidos son: las vistas abstractas de datos, los diagramas de configuración, y los mapas ADV.

Implementación

La implementación traduce los objetos de interfaz a objetos de implementación, y puede incluir arquitecturas elaboradas (por ejemplo, cliente-servidor), en las cuales las aplicaciones son clientes de un cliente de un servidor de datos que contiene los objetos conceptuales. En la medida que la arquitectura de la aplicación (por ejemplo un conjunto de páginas HTML sin ninguna estructura relacional subyacente) no se adapte a un modelado orientado a objetos, el uso de la metodología y su traducción a la aplicación final puede ser más o menos dificultoso.

3.2.6 Conclusiones

Como hemos podido ver la naturaleza heterogénea de los sistemas de representación hipermedia (comentada en el Capítulo 2) se traduce en multitud de opciones a la hora de traducir la representación proporcionada por los sistemas de representación a la aplicación ejecutable.

Respecto a los sistemas de representación que no inducen ni metodología ni disponen de un entorno de desarrollo cabe comentar que en principio parece bastante poco razonable su uso, ya que según [Barry 01] la falta de un entorno integrado dificulta el uso de este tipo de formalismos

Por otro lado, la mayor ventaja de disponer de un entorno de diseño-desarrollo es que a partir de descripciones de alto nivel (como son los componentes Amsterdam o las redes de Petri) se pueden obtener aplicaciones hipermedia. El mayor inconveniente es que las aplicaciones que se obtienen están restringidas a la potencia expresiva proporcionada por los respectivos sistemas de representación. La aproximación seguida en esta tesis intentará conseguir las ventajas evitando los inconvenientes. Con este fin proporcionará un lenguaje basado en XML para representar el modelo hipermedia, y utilizando su semántica de presentación predeterminada generar automáticamente prototipos. Para evitar los inconvenientes de esta aproximación cerrada se permite la inclusión de procesos subordinados (clases Java) que son invocados dinámicamente desde la aplicación hipermedia. Además el uso de estas técnicas se restringe a conceptualización y prototipado, dejando abiertas las fases de diseño y conceptualización.

Finalmente hemos podido comprobar como la existencia de un sistema de representación hipermedia bien definido permite definir y guiar una metodología de desarrollo. Dicha metodología únicamente ha de encargarse de proporcionar representaciones de las estructuras de modelado propuestas por los sistemas de representación hipermedia para construir la aplicación. Basándonos en esta idea, hemos fusionado un modelo hipermedia de diseño propio (Pipe) con un modelo de proceso aceptado por la comunidad hipermedia [Fraternali 98]. De esta forma vamos a guiar las etapas de conceptualización y prototipado a través de las representaciones proporcionadas por el modelo Pipe. Una característica de esta aproximación es que tanto las ventajas e inconvenientes de los sistemas de representación son asimiladas en su totalidad por las metodologías definidas a partir de ellos.

3.3 Documentos, Transformaciones, y Componentes (DTC)

3.3.1 Introducción

La aproximación DTC, *Documentos, Transformaciones, y Componentes*, es una metodología para la construcción de aplicaciones software genéricas basada en la utilización de documentos XML, transformaciones XSLT, y componentes software reutilizables [Sierra 99], [Sierra 00a], [Sierra 00b], [Sierra 01]. Aunque por naturaleza no es específica para aplicaciones hipermedia, puede utilizarse, y de hecho ha sido utilizada con este fin [Navarro 00c]. La inclusión en este capítulo proporciona otra visión alternativa a aproximaciones como SMIL o la presentada en el Capítulo 6 de esta tesis. En efecto, si bien en estas aproximaciones los lenguajes de marcado se utilizan como un mecanismo para *inicializar* un sistema hipermedia, y en base a dicha inicialización obtener una aplicación hipermedia, la aproximación DTC es bastante distinta. DTC utiliza lenguajes de marcado para *construir* una aplicación mediante descripciones de alto nivel, que se combinan a través de transformaciones XSLT. La construcción de dicha aplicación se produce mediante la combinación de estos lenguajes, que en último término tienen componentes software asociados. Es la unión de estos componentes software los que realmente proporcionan el código de la aplicación que estamos construyendo.

3.3.2 La aproximación DTC

Existen dominios donde una parte fundamental del proceso de desarrollo es proporcionar los contenidos a procesar por parte de la aplicación. Ejemplos de tales dominios son los sistemas basados en el conocimiento [Studer 99], o el dominio educativo [Fernández 97b]. En estos casos la existencia de lenguajes cercanos a las intuiciones de los expertos en el dominio, posiblemente sin vinculación al desarrollo software, son una parte crítica para el éxito del proyecto. DTC considera por tanto que los *lenguajes específicos del dominio*, DSL [Walton 96], son cruciales para el desarrollo con éxito de este tipo de aplicaciones basadas en contenidos. [Studer 99] proporciona una buena definición de DSL: "un pequeño, normalmente declarativo, lenguaje expresivo sobre las características fundamentales de un conjunto de programas en un dominio de problema particular". Para la descripción del contenido DTC utiliza un significado descriptivo, orientado al dominio, con independencia de las interpretaciones operacionales concretas inducidas por usos concretos de tales lenguajes [Fuchs 97].

DTC hace un uso extensivo de los principios DSL. La construcción de aplicaciones según esta metodología puede interpretarse como el proceso de formulación, transformación, composición y operacionalización de lenguajes específicos del dominio. A alto nivel, DTC divide el desarrollo de la aplicación en dos actividades fundamentales. La primera consiste en proporcionar una descripción explícita de la aplicación en un DSL adecuado. La segunda consiste en proporcionar soporte operacional para este DSL. Además, durante la descripción de la aplicación, DSL promueve una separación explícita entre la *descripción de contenido* y la *descripción software*.

Los contenidos se describen en términos de uno o varios DSLs denominados *DSLs de contenidos*. Estos DSLs intentan ser lo más neutrales respecto a un uso concreto, centrándose en los aspectos estructurales de la información en el dominio de aplicación, en vez de preocuparse de cómo va a ser procesada la información. Por tanto, los DSLs pueden ser formulados en términos familiares para los expertos del dominio que proporcionan los contenidos. De esta forma, las descripciones que se ajustan a estos DSLs pueden ser reutilizados de múltiples formas, bien en la misma aplicación, o bien en otra aplicación del mismo dominio. Sin embargo, para permitir usos particulares de un contenido concreto, es necesario proporcionar información contextual (o dependiente del uso) adicional. Por ejemplo, en una aplicación de redes de transporte, podemos tener un DSL para describir los datos relevantes del dominio (por ejemplo estructura y tiempos) de la red de transporte. Pero a este nivel es importante la inclusión de datos adicionales para describir información presentacional (por ejemplo, coordenadas, fuentes y colores) que serán necesarias para visualizar la red de transporte. DTC contempla la descripción de este tipo de información proporcionando DSLs de contenidos adicionales, denominados *DSLs de contenidos dependientes del uso*.

Una vez proporcionada la descripción de los contenidos, se debe determinar los usos de dichos contenidos dentro de la aplicación. Con este fin se deben seleccionar DSLs adecuados orientados a como deben *interpretarse* los contenidos dentro de la aplicación, los *DSLs de interpretación*. El enlace entre los DSLs de contenidos y de interpretación se lleva a cabo mediante *transformaciones* escritas por los desarrolladores. Por ejemplo, la búsqueda de rutas de transporte en una red de metro según un criterio dado puede ser fácilmente reformulado en la tarea de buscar caminos en un grafo dirigido y con pesos. De esta forma, el DSL que describe las redes de transporte puede interpretarse en términos de un lenguaje para describir grafos dirigidos y con pesos. La interpretación real se lleva a cabo escribiendo una transformación del DSL de redes de transporte en el DSL de descripción del grafo.

Otros aspectos de la aplicación, no directamente relacionados con la interpretación de los contenidos, deben describirse en términos de *DSLs de la aplicación* adicionales. Estos DSLs permiten la descripción de otros aspectos tales como la estructura de la interfaz gráfica de usuario, la interacción del usuario, y el control. Los DSLs de interpretación son en último término un caso particular de DSLs de la aplicación. Finalmente, la composición adecuada de los DSLs de la aplicación produce un único DSL de descripción de la aplicación. La propia aplicación está descrita en términos de este DSL.

DTC utiliza actualmente XML para definir los DSLs. De esta forma, cada DSL está concebido como un lenguaje de marcado XML. Así disponemos del marco sintáctico común para representar de una forma estructurada toda la información que necesitamos para el desarrollo de la aplicación, junto con un conjunto de tecnologías asociadas que hacen más sencillo del desarrollo de aplicaciones basadas en lenguajes específicos del dominio independientes de plataformas de implementación particulares. Sin embargo, para algunos dominios de aplicación, el uso directo de documentos estructurados XML podría ser un inconveniente (por ejemplo, supongamos el proporcionar los datos sobre la red entera de transporte en XML). Por tanto DTC también introduce el concepto de *autoría del documento* en términos de sintaxis más adecuadas. De esta forma se pueden configurar *marcos de autoría* genéricos (por ejemplo, editores visuales) para definir *sintaxis de autoría* junto con relaciones de traducción entre estas sintaxis y los lenguajes de marcado específicos basados en XML.

Por último, está la semántica asociada los DSLs de la aplicación. DTC se ocupa primordialmente de la *semántica operacional*. De esta forma, cada DSL de la aplicación debe estar acompañado de un conjunto de artefactos computacionales (es decir, componentes) proporcionando uno (o un conjunto) de significado(s) operacional(es) a la información que se ajusta a este lenguaje. En lo referente a los DSLs de contenidos, DTC no prescribe ningún método particular para describir la semántica asociada. De esta forma podría proporcionarse un documento informal, o establecerse mediante alguna técnica formal. De hecho, cuando se definen las transformaciones entre DSLs de contenidos y DSLs de interpretación, la semántica operacional de

los lenguajes interpretados podría atribuirse automáticamente a los contenidos. La Figura 3.12 ilustra las principales actividades que se llevan a cabo durante el desarrollo de una aplicación según DTC.

En esta figura podemos observar como se proporcionan DSLs adecuados a partir de los requisitos de la

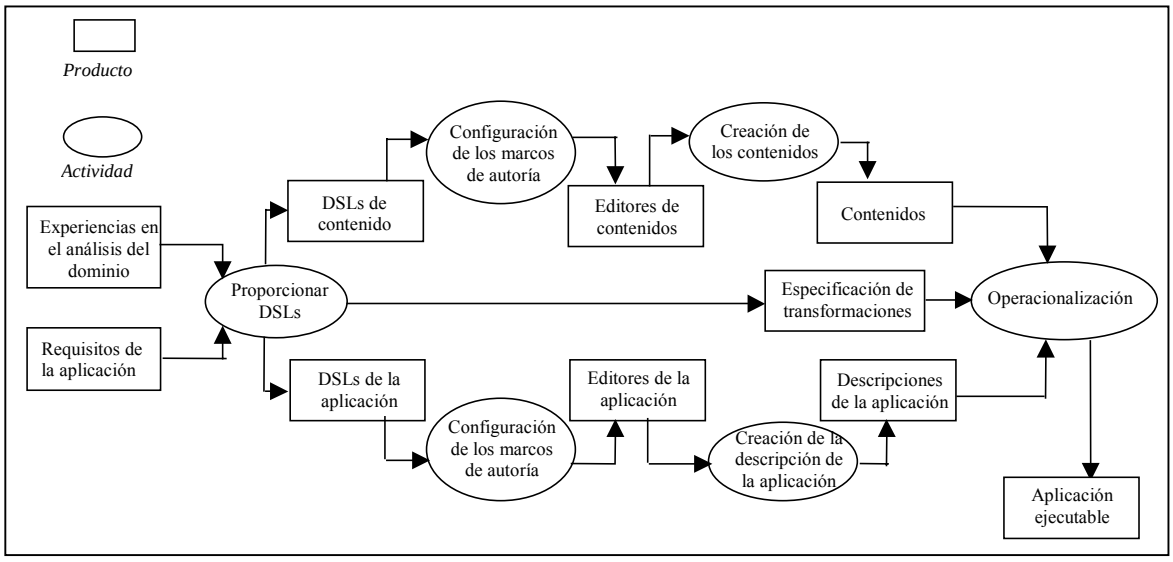

Figura 3.12 Principales productos y aplicaciones involucrados en el desarrollo de una aplicación según DTC.

aplicación y de las experiencias en el análisis del dominio. Estos DSLs sirven para la descripción de los contenidos y otros aspectos de la aplicación. Además, también se proporcionan transformaciones para relacionar los contenidos con los DSLs de interpretación. La definición de sintaxis de autoría adecuadas permite la configuración de los marcos que producen editores adaptados a la creación de contenidos y de la descripción de la aplicación. Finalmente, las descripciones de contenidos, especificaciones de transformación, y descripciones de la aplicación son la entrada de un proceso operacional que produce la aplicación ejecutable.

3.3.3 Construcción de aplicaciones operacionalizando las descripciones DTC

En la construcción de la instancia de una aplicación DTC hay dos fases fundamentales: la *fase de autoría* y la *fase de operacionalización*. En la primera fase se proporcionan descripciones de las aplicaciones. Con este fin, se crean los contenidos para después transformarlos en lenguajes de interpretación. En esta fase también se proporcionan otros aspectos de la aplicación. Finalmente se componen las descripciones de la aplicación y las interpretaciones derivadas en una única descripción de la aplicación. En la fase de operacionalización se obtiene el programa ejecutable de la descripción de la aplicación y de un conjunto de componentes. Los ingredientes de esta fase son: la descripción de la aplicación proporcionada en la fase de autoría, *componentes software* adecuados que proporcionan el soporte operacional a los DSLs utilizados, y finalmente un *repositorio DTC* que agrupa todos los componentes requeridos. Veamos en detalle la fase de operacionalización.

3.3.3.1 Los ingredientes de la operacionalización

La operacionalización de las descripciones de aplicación DTC puede verse como el proceso de proporcionar artefactos computacionales con un comportamiento consistente con dichas descripciones. Por *significado* estamos suponiendo *significado operacional*. Ya que DTC no compromete ningún lenguaje de descripción particular, el proceso de operacionalización debe ser necesariamente abierto. De esta forma no tiene sentido la existencia de un motor de operacionalización universal, que dada cualquier descripción de aplicación DTC proporcione el artefacto computacional correspondiente. Por otro lado, DTC fomenta el desarrollo de las descripciones mediante una apropiada combinación de lenguajes diseñados para describir cada aspecto de la aplicación. De esta forma, los artefactos computacionales asociados con las aplicaciones concretas pueden construirse mediante la composición adecuada de artefactos más simples asociados a lenguajes utilizados en la descripción. Por tanto, en DTC, la composición de lenguajes tiene una contrapartida operacional en la composición de componentes software.

Entre estos componentes podemos encontrar los componentes *atómicos* y los *combinadores*. Los primeros llevan a cabo totalmente sus tareas sin necesidad de utilizar más componentes. Los segundos son aquellos que requieren la ayuda de otros componentes para llevar a cabo su funcionalidad. En ambos casos, los componentes trabajan sobre la representación DOM de los documentos.

Una vez disponibles un repertorio adecuado, y una descripción de la aplicación, los componentes deben ser correctamente instanciados, y sus instancias ensambladas para obtener la aplicación ejecutable. Otra vez la consideración de un "ensamblador universal" está fuera de discusión, ya que en ensamblaje depende en gran medida de la semántica particular de cada DSL, y esta semántica está confinada en los componentes asociados a cada lenguaje. De esta forma, la idea clave es permitir que los componentes controlen ellos mismos el proceso de operacionalización. Así los componentes deben ser capaces tanto de procesar descripciones que se ajusten a sus lenguajes asociados como de delegar el procesamiento en otros componentes cuando encuentre partes de descripciones fuera de su ámbito. Con este fin se utiliza el repositorio DTC. El siguiente apartado profundiza en detalles.

3.3.3.2 El proceso de operacionalización

La operacionalización DTC es un proceso cooperativo entre los componentes asociados con los DSLs compuestos en el lenguaje de descripción de la aplicación. De esta forma los componentes trabajan sobre el árbol del documento asociado con la descripción total de la aplicación. Para permitir esta colaboración dichos componentes utilizan un repositorio DTC común.

El proceso de operacionalización empieza interrogando al repositorio DTC con un *contexto inicial*. Por *contexto* entendemos el árbol de la instancia del DSL de descripción de la aplicación junto con una referencia a uno de sus nodos. Este contexto inicial normalmente se refiere al nodo asociado con el elemento documento.

El comportamiento del repositorio DTC es muy simple. Cuando recibe un contexto comprueba si hay instanciado un componente como resultado de procesar este contexto. Si es así, el repositorio devuelve la instancia resultante. Si no, interroga a los componentes disponibles hasta que encuentre uno capaz de procesar el contexto. Con este fin, es necesario que los componentes dispongan de un interfaz para decidir a que contextos pueden aplicarse. Las implementaciones concretas de este interfaz no son relevantes para el proceso, pero podría ser tan simple como comprobar la etiqueta y/o el espacio de nombres [W3C namespaces] asociados con el nodo referenciado por el contexto, o tan complejo como validar el documento con raíz en el nodo contra la gramática del lenguaje soportado por el componente. Cuando el repositorio encuentra un componente capaz de ser aplicado sobre el contexto, la instancia, guarda la instancia como asociada en el procesamiento del componente, y delega el procesamiento a esta instancia. La Figura 3.13 ilustra este proceso.

En dicha figura podemos ver como el proceso empieza con una pregunta al repositorio DTC mediante un contexto para la raíz del árbol de descripción. Entonces, el repositorio selecciona el componente adecuado para procesar este contexto (etiquetado por una circunferencia). El siguiente paso para el repositorio es instanciar el componente (al cual llamaremos I_0) y delegar en él el procesamiento del contexto. Después de un cierto procesamiento, I_0 decide preguntar al repositorio para el procesamiento del contexto etiquetado con un trapecio. El repositorio se comporta como acabamos de describir, seleccionando e instanciando el componente correspondiente (llamemos I_1 a la nueva instancia), y delegando el comportamiento del contexto en dicha instancia. Eventualmente I_1 termina el procesamiento sin más interacciones y devuelve el control al repositorio, el cual se lo devuelve a I_0 (junto con I_1). I_0 establece algún tipo de relación con I_1 y continúa el procesamiento de su contexto hasta que decide preguntar otra vez al repositorio (esta vez con el contexto etiquetado con un cuadrado sin cima). Esto conduce a la creación de una nueva instancia (I_2) que se encarga del procesamiento del contexto. I_2 termina la tarea sin más interacciones, así que el control se devuelve al repositorio, y de ahí a I_0, que después de establecer la correspondiente relación con I_2, termina el procesamiento. El control se devuelve al repositorio, que devuelve I_0 (y por tanto el artefacto ensamblado) como resultado del proceso de operacionalización.

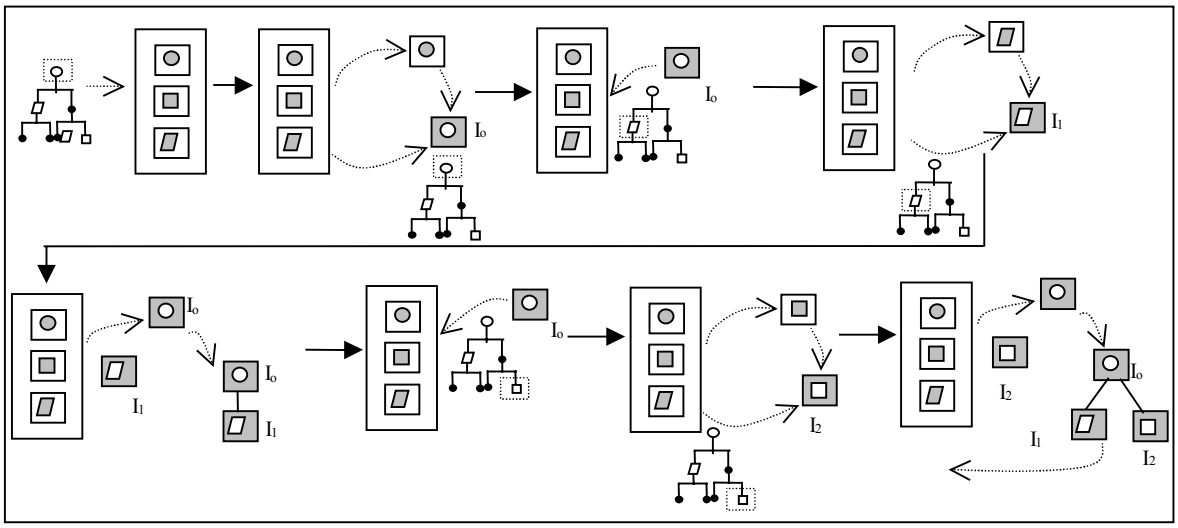

Figura 3.13 Ejemplo del comportamiento del proceso de operacionalización DTC.

Aunque este es el comportamiento estándar del proceso de operacionalización, en la práctica pueden utilizarse otros más complejos. Su estudio está fuera del alcance de este trabajo. En [Sierra 01] pueden encontrarse dichos comportamientos.

3.3.4 Conclusiones

DTC es un proceso de operacionalización flexible para el desarrollo de aplicaciones basadas en contenidos que utilizan tecnologías de marcado generalizadas. El principal beneficio de este proceso es proporcionar una clara separación entre la descripción de la aplicación y las tecnologías de implementación. Este aspecto es crucial en el desarrollo de aplicaciones basadas en contenidos, donde la mantenibilidad y portabilidad son factores clave para el éxito final del proyecto.

La utilización de la descripción de la aplicación a un nivel lingüístico como una etapa independiente de la operacionalización mejora la mantenibilidad, ya que podemos razonar a un nivel mayor que el proporcionado por el código de los componentes que se ensamblan. Además muchos de los cambios y modificaciones en la aplicación se producen en el nivel de documento, sin esfuerzo de programación. Por otro lado, la programación basada en componentes favorece el mantenimiento automático de la aplicación. Los documentos de contenidos y los componentes pueden ser reutilizados en diversas aplicaciones.

DTC también mejora la portabilidad, ya que las descripciones son más independientes de tecnologías concretas de implementación.

Por otro lado a pesar de la aparente similitud entre DTC y PlumbingXJ (que presentaremos en el Capítulo 6), las aproximaciones son sustancialmente distintas. Para empezar DTC es una metodología de desarrollo de propósito general, mientras que PlumbingXJ es un modelo de proceso que fija unas técnicas de prototipado para aplicaciones hipermedia. DTC busca la *construcción* de aplicaciones a partir de descripciones parciales de la misma que se unifican en un único documento, mientras que PlumbingXJ *inicializa* una aplicación (GAP, que en última instancia es un sistema hipermedia) con distintos interfaces gráficos y contenidos. Esta última característica hace que DTC provoque el ensamblaje de diversos componentes software, mientras que en PlumbingXJ hay un único programa que procesa al documento. Si optamos por considerar PlumbingXJ como un caso particular de DTC, entonces cualquier aplicación que procese un documento XML sería una aplicación DTC. No debe deducirse de este párrafo una comparación crítica entre aproximaciones para demostrar cual es la más ventajosa. El razonamiento simplemente pone de manifiesto las diferencias entre ambas aproximaciones.

Es importante remarcar que DTC es una tecnología en desarrollo, y que muy posiblemente futuras versiones de la misma difieran de lo aquí presentado, pero el núcleo fundamental permanecerá estable.

3.4 Conclusiones

El título de este capítulo bien podría haber sido *Miscelánea*. En él hemos revisado varias aproximaciones a la hora de construir una aplicación hipermedia. Decimos varias y no todas, ya que de haber procedido a un estudio de todas las técnicas de desarrollo existentes, la extensión en páginas de dicho capítulo hubiera sido cercana a $\aleph_0$.

En la primera parte estudiábamos diversos modelos de proceso disponibles a la hora de construir una aplicación hipermedia. Aunque en trabajos previos nosotros presentábamos nuestro propio modelo de proceso [Navarro 01a], debido a la similitud del propuesto por Fraternali (y por extensión al propuesto por Ginige y Lowe) optamos por elegir aquel aceptado por la comunidad hipermedia. Básicamente es un modelo de proceso evolutivo que introduce un bucle previo de conceptualización y prototipado. La necesidad de este bucle radica en la especial importancia que tienen ahora tanto el cliente como la interfaz gráfica de usuario de la aplicación (esquema navegacional nos parece un término más adecuado).

En la segunda parte vimos una serie de metodologías de construcción de aplicaciones hipermedia basadas en los sistemas de representación hipermedia presentados en el Capítulo 2. Hay dos ideas fundamentales que debemos extraer de esta parte. La primera, una semántica de presentación por defecto facilita la construcción automática de aplicaciones hipermedia en base a representaciones de alto nivel de la misma. La segunda, el disponer de un sistema de representación hipermedia permite definir una metodología de desarrollo basada en su totalidad en dicho sistema. En esta tesis presentaremos un modelo hipermedia con semántica de presentación por defecto que guiará en su totalidad las fases de conceptualización y prototipado del modelo de proceso de Fraternali.

Hemos de comentar un par de cuestiones sobre este hecho. Primero, hemos decidido centrarnos exclusivamente en las fases de conceptualización y prototipado para evitar el diseño de aplicaciones hipermedia limitado por la potencia expresiva del sistema de representación hipermedia que guía a dicha metodología. Segundo, pensamos que en el desarrollo anterior al no determinar la construcción de la aplicación final, no variamos la naturaleza de modelo de proceso de la aproximación. De todas formas esta es una cuestión meramente terminológica que en principio no es fácil de determinar y puede suscitar polémica. Por último, la necesidad de un nuevo sistema de representación hipermedia, y de las ventajas que se derivan de su integración con el modelo de proceso de Fraternali pueden encontrarse en los capítulos 5 y 6 de esta tesis.

Finalmente hemos presentado la metodología de construcción de aplicaciones DTC (el porque puede encontrarse en la introducción de este capítulo). Su probada viabilidad en el dominio hipermedia la hace obligada candidata a la construcción de este tipo de aplicaciones. Como mayor ventaja de esta aproximación cabe citar su naturaleza abierta, que evita el problema de restringir la aplicación generada a la potencia expresiva de un sistema de representación hipermedia concreto. Esta a su vez es su mayor inconveniente, ya que en última instancia la convierte en una técnica de programación de alto nivel (posiblemente, y aunque en un área de aplicación distinta, un lenguaje de cuarta generación), poco adecuada para labores de conceptualización y prototipado. Por lo tanto el dominio de aplicación idóneo de DTC es la construcción real de la aplicación final, pudiéndose utilizar como técnica concreta de implementación dentro del modelo de proceso PlumbingXJ.

4 Lenguajes de marcado

4.1 Introducción

En este capítulo presentaremos una visión general sobre los lenguajes de marcado estándar (SGML y XML), así como de las tecnologías asociadas a estos lenguajes. La razón de incluir un capítulo de estas características se debe a dos motivos fundamentales. La primera es estudiar los estándares (o *recomendaciones*, utilizando la terminología W3C. A lo largo de este capítulo utilizaremos ambos términos indistintamente) que han influido e influyen notablemente en el campo de las aplicaciones hipermedia. Entre los más destacables cabe citar, HyTime, HTML, SMIL, o XLink. La segunda se debe al uso de estas tecnologías que se va a realizar en el modelo de proceso PlumbingXJ. En particular dicho modelo contará con la potencia expresiva de XML, y la potencia referencial de XPath para construir descripciones de aplicaciones hipermedia a partir de las cuales generar automáticamente prototipos. Aunque en este modelo se utiliza XPath, como veremos, es posible utilizar otros estándares como XSLT o XPointer. Por esa razón los estudiaremos brevemente.

Debemos destacar el increíble auge en los últimos años de las tecnologías de marcado generalizadas, y en concreto de XML. Fruto de esta expansión es una ingente cantidad de estándares o pseudoestándares instalados en todas las áreas de la informática. Eligiendo un grupo de ellos al azar (de entre los no tratados en este capítulo) podemos encontrarnos por ejemplo, con el *Election Markup Language* [EMLb], para el intercambio estructurado de información entre proveedores de soporte hardware, software, y/o servicios; el *Vocabulary Markup Language* [VocML], para la representación estructurada de recursos de sistemas y servicios de organización del conocimiento; o el *Petri Net Markup Language* [PNML], lenguaje para el intercambio de redes de Petri. Hacer un estudio, o mención de todos los estándares disponibles sería una tarea compleja y cuanto menos estéril. Compleja debido a la cantidad de los mismos, a su imparable ritmo de aparición, y a su falta de estabilidad (es tan grande el número de lenguajes basados en tecnologías de marcado generalizado que incluso hay siglas que se repiten. Así EML no solo son las siglas del *Election Markup Language*, sino también las del *Educational Modelling Language* [EMLa]). Estéril porque muchos de estas aplicaciones no tienen ninguna relación con el trabajo desarrollado en esta tesis. De todas formas las siguientes direcciones de Internet: *Las páginas XML de Robin Cover* (*http://www.oasis-open.org/cover/)*, el sitio Web oficial del *World Wide Web Consortium* (*http://www.w3.org)*, el *Portal de la Industria XML* (*http://www.xml.org/xml/news_market.shtml)*, o el sitio Web de *O`Reilly* (*http://www.xml.com/*) son valiosas y actualizadas fuentes de información sobre XML y sus estándares asociados.

Por lo tanto en este capítulo veremos una descripción del Lenguaje de Marcado Generalizado Estándar (SGML), el Lenguaje de Marcado Extensible (XML), el lenguaje HyTime, una serie de aplicaciones de estos lenguajes que han influido notablemente en el dominio hipermedia y en el campo de la generación automática de interfaces de usuario, y por último una serie de estándares asociados a XML. No se proporcionará una visión en profundidad en ninguno de estos apartados, ya que no pensamos que esa no debe ser la tarea fundamental de esta tesis. Además al ser todos ellos estándares documentados, se dispone en cada caso de una referencia valiosísima a la hora de obtener una descripción de los mismos. Finalmente, la aplicación de los lenguajes de marcado en el ámbito concreto de esta tesis puede encontrarse en el apartado 6.3 del Capítulo 6.

4.2 Lenguajes de marcado, el lenguaje de marcado generalizado estándar (SGML) y el lenguaje de marcado extensible (XML)

4.2.1 Lenguajes de marcado

En este apartado proporcionaremos una visión global de las características fundamentales de SGML y XML, prestando mayor atención a SGML, al englobar este todas las capacidades de XML. Antes de nada deberíamos definir que es un lenguaje de marcas (o de marcado). Según [Hopcroft 79] un *lenguaje* (formal) es "un conjunto de cadenas de símbolos de algún alfabeto". Según [Goldfarb 90] *marcado* se define como "texto que se añade a los datos de un documento con el fin de añadir información sobre él". Por tanto si juntamos ambas definiciones, un *lenguaje de marcado* sería un "conjunto de cadenas de símbolos de algún alfabeto que se añade a los datos de un documento con el fin de añadir información sobre él". Esta es (en nuestra opinión) la

mejor definición de alto nivel que se puede dar de un lenguaje de marcas, con independencia de cuestiones que puedan surgir como ¿cuál es la manera de distinguir el lenguaje de marcado del propio documento? O ¿por qué utilizar lenguajes de marcado?

Para responder a esta última pregunta retornaremos a la definición de lenguaje de marcado. La idea clave radica en "añadir información al documento". En efecto, en el tratamiento "clásico de lenguajes formales" [Aho 90] a partir de los terminales del lenguaje que es objeto de procesamiento, se puede identificar una estructura sintáctica que permite especificar un procesamiento dirigido por dicha sintaxis. Por ejemplo, la palabra reservada `if`, permite reconocer en los lenguajes de programación la raíz de una estructura sintáctica que sirve para procesar una orden condicional.

En el mundo de los lenguajes de marcado los lenguajes objeto de tratamiento (aquellos a los que se añaden las marcas) no permiten *per se* identificar una estructura sintáctica sobre la cual basar su procesamiento. El problema radica en que la mayor parte de las veces el lenguaje objeto es *lenguaje natural*. En este caso, la única estructura sintáctica que se puede identificar es la del propio idioma al que pertenecen las sentencias del lenguaje, y esta sintaxis, la mayoría de las veces, no permite deducir el procesamiento que se desea (evidentemente, salvo tratamientos explícitos del lenguaje natural). Es por esta razón que debemos imponer las marcas sobre el lenguaje objeto de tratamiento, es decir, información sobre la información, o *metainformación*. Con esta aproximación queda bastante claro las áreas de aplicación de los lenguajes de marcado (por lo menos en los inicios): añadir marcas a documentos (en la "acepción editorial") para facilitar su tratamiento. Nótese que los documentos "tradicionales" (libros, manuales, folletos, etc.) están formados por una serie de sentencias en lenguaje natural. El tamaño, color o forma de la fuente puede indicar al lector información sobre el propio documento (título, autor, sección, etc.), pero una computadora es ajena a este tipo de información para tratar el documento (a no ser que la propia tipografía se considere una marca). Es por este hecho que se hace necesario indicar explícitamente esta información.

A partir de esta idea original han ido apareciendo variaciones, que respetan más o menos este concepto de lenguaje de marcado. El uso más alejado de la concepción original de los lenguajes de marcado, y que solo se justifica desde la legibilidad proporcionada por este tipo de formalismos, lo representan aproximaciones que utilizan a los lenguajes de marcas como lenguajes de *script* [Shirota 01]. En esta tesis no se va a justificar o criticar dicha aproximación, pero en principio no es la utilizada en esta tesis. De hecho el uso que se va a hacer de los lenguajes de marcado será el de estructurar contenidos, y el de estructurar el esquema navegacional de una aplicación hipermedia utilizando este tipo de lenguajes. El primer uso entra de lleno en la aproximación anteriormente presentada. El segundo, aunque se aleja de dicha aproximación, aprovecha las facilidades de legibilidad de XML para describir esquemas navegacionales y de sincronización de distintos medios, al igual que hacen estándares como SMIL o HyTime. Esta es una tercera vía intermedia entre la estructuración de contenidos, y el uso de etiquetas como lenguaje de *script*, que utiliza a los lenguajes de marcado para la especificación de estructuras. Dichas estructuras pueden ser de datos, como en el PNML, de interfaces, como el UIML [Abrams 00], o de presentaciones multimedia, como el SMIL.

4.2.2 SGML

SGML y su relación con los lenguajes formales clásicos

El *Standard Generalized Markup Language*, o SGML [ISO SGML], es una evolución del lenguaje de marcado GML desarrollado en los años ochenta por C.F. Goldfarb en IBM [Goldfarb 90]. A pesar de aparecer la denominación "lenguaje de marcado" en su composición, SGML no es un "conjunto de cadenas de símbolos de algún alfabeto que se añada a los datos de un documento con el fin de añadir información sobre él". SGML es un conjunto de convenciones sintácticas y reglas añadidas que permiten definir lenguajes de marcas. Es decir, SGML es a los lenguajes de marcas lo que EBNF a los lenguajes de programación. Además, los lenguajes de marcas que podemos definir a través de SGML van más allá de la definición proporcionada hasta ahora en esta tesis. Una definición más acorde de los lenguajes de marcas definidos mediante las facilidades proporcionadas por SGML sería: "conjunto de marcas jerarquizadas y atribuidas que se añaden a los datos de un documento con el fin de capturar su estructura (según algún criterio definido), añadiendo información sobre él".

Antes de analizar la definición anterior analicemos que es una "marca". Una marca es un elemento del lenguaje de marcas. Es decir, una "palabra del lenguaje de marcas" utilizando la terminología de [Hopcroft 79], o un "terminal del lenguaje de marcas" utilizando la terminología de [Aho 90]. Estas dos últimas

caracterizaciones aunque correctas pueden llevar a confusión si intentan aplicarse directamente a las producciones o reglas gramaticales de SGML. Por ejemplo si consideramos las producciones SGML:

```
<!ELEMENT a - - (b)>
<!ELEMENT b - - EMPTY>
```

El lenguaje que genera es:

```
<a><b></b></a>
```

Es decir, las marcas o terminales serían <a>,, , . Este es un ejemplo "sesgado" de uso de SGML, ya que estas marcas no estructuran ninguna información. Si consideramos:

```
<!ELEMENT a - - (b)>
<!ELEMENT b - - (#PCDATA)>
```

Una de las sentencias del lenguaje generado sería:

```
<a><b>Hola Mundo</b></a>
```

Estrictamente hablando, y supuesto que hubiéramos definido unas reglas de derivación para las producciones SGML, los cinco terminales del lenguaje son en este caso: <a>, , , , Hola Mundo. Nótese que al igual que con los *identificadores* en lenguajes de programación, #PCDATA no identifica a un solo terminal, sino a un conjunto de terminales (en este caso, los *datos* del documento, en contraposición a las *etiquetas*, representadas por el resto de terminales). La cuestión radica en diferenciar los *documentos generados* mediante las producciones SGML (etiquetas y datos), y el *lenguaje de marcas generado* mediante las producciones SGML (etiquetas), que en el ejemplo anterior es una etiqueta a, que puede contener únicamente una etiqueta b, la cual puede contener datos carácter. Es decir, aunque SGML sea un lenguaje para definir lenguajes de marcas, en la práctica define tanto al lenguaje de marcas, como a los datos (representados por los contenidos #PCDATA, RCDATA, y CDATA) que va a estructurar dicho lenguaje de marcas.

Como ya comentamos antes, la utilidad de las marcas insertadas en el lenguaje objeto es la de proporcionar terminales en el lenguaje que permitan identificar una sintaxis sobre la que definir un tratamiento. En efecto, supongamos que tenemos la frase:

```
El niño corre
```

Esta es una sentencia del lenguaje natural. Podemos considerarla como una frase generada por la gramática:

```
S-> SN SV
SN -> DET NOM
DET -> el | la | un ...
NOM -> niño | niña | perro ....
SV -> VERBO COMP
VERBO -> correr | corre | corriendo | saltar ..
COMP -> ε | rápido ...
```

Donde las palabras en mayúsculas denotan no terminales y las palabras en minúscula denotan terminales. En dominios restringidos podríamos proporcionar caracterizaciones similares (incluso aumentada con atributos para garantizar las restricciones contextuales de género y número), pero la mayoría de las veces esta estructura sintáctica no nos permitirá (o sería extremadamente costos) especificar el tratamiento deseado (a no ser, claro, que dicho tratamiento se base en la sintaxis del lenguaje natural).

Por ejemplo, consideremos la frase del lenguaje natural,

```
Antonio Navarro. Una autobiografía de Antonio Navarro. Antonio Navarro.
```

Si la estructura o información que nos interesa capturar es que en una descripción aparece el nombre de un libro y su autor, necesitamos indicar esta información explícitamente (como podemos ver en el ejemplo, ni los puntos, ni los nombres propios sirven para identificar dicha estructura). Además debemos indicar donde puede o debe aparecer la información. Las producciones SGML:

```
<!ELEMENT desc - - (libro, autor)>
<!ELEMENT libro - - (#PCDATA)>
<!ELEMENT autor - - (#PCDATA)>
```

permiten marcar la información anterior de la siguiente manera:

```
<desc>
  <libro> Antonio Navarro. Una autobiografía de Antonio Navarro.</libro>
  <autor> Antonio Navarro.</autor>
</desc>
```

Es decir, la utilidad de SGML es la de introducir una serie de marcas que permitan identificar la siguiente estructura sintáctica:

```
S->DESC
DESC-> <desc>LIBRO AUTOR</desc>
LIBRO -> <libro>D</libro>
AUTOR -> <autor>D</autor>
D -> Σ*
```

con el fin de poder especificar un tratamiento del lenguaje original basado en la sintaxis identificada a partir de los terminales representados por las marcas del lenguaje de marcas. Nótese que Σ^* representa a los datos del documento (de los cuales debería excluirse el carácter < para poder identificar el comienzo de la etiqueta que denota el final del elemento). En este contexto, el lenguaje de marcas generado está formado por todos aquellos terminales del lenguaje que no derivan de la producción:

```
D -> Σ*
```

Es decir, está formado por las etiquetas, y no por los datos. Por supuesto, SGML añade la potencia de los atributos, y restricciones tales como la unicidad del valor de atributos de tipo ID, así como otras facilidades, pero en la práctica, y desde el punto de vista del autor de esta tesis, SGML no deja de ser una sintaxis alternativa a EBNF fruto de la evolución del lenguaje GML por parte de Goldfarb y su equipo.

Por tanto, y desde el enfoque de [Ullman 79], y [Aho 90] podríamos decir que el fin último de SGML es hacer explícita la secuencia de derivaciones en una sentencia del lenguaje de los datos. Precisamente son las marcas del lenguaje de marcas las que llevan cuenta de esta secuencia de derivaciones, proporcionando al lenguaje origen la estructuración suficiente para especificar un procesamiento basado en la sintaxis.

Principios de uso de SGML

SGML se basa en tres principios de uso:

- Independencia entre la estructura y el contenido de un documento.
- Independencia entre la estructura y el tratamiento de un documento.
- Independencia de la plataforma.

Estos principios representan una serie de guías no vinculantes que deberían tenerse en cuenta a la hora de aplicar SGML. La primera hace patente que SGML es un lenguaje específicamente diseñado para capturar la estructura de un documento general (o de un tipo de documento), con independencia de los contenidos concretos que cada documento pueda tener. En efecto, un *fax* tiene una estructura bien definida sin importar quien sea la persona que lo remite cada vez, o el mensaje incluido.

El segundo punto, también conocido por *marcado declarativo* o *descriptivo*, representa a la vez la mayor ventaja y el mayor inconveniente de SGML. Cuando utilizamos la etiqueta HTML (lenguaje SGML) `<b>` con el fin de indicar que su contenido debe aparecer en negrita, estamos violando la segunda norma, ya que estamos ligando la estructura identificada por la etiqueta (texto que debe aparecer resaltado del resto) con el tratamiento aplicado a la misma (para resaltar el texto, aparecerá en negrita). SGML predica que la identificación de la estructura debe ser totalmente independiente del tratamiento que se quiera dar a esta. Así, en vez de utilizar la etiqueta `<b>`, deberíamos utilizar una etiqueta `<importante>`, y en un segundo paso asignarle un tratamiento. Por ejemplo que apareciera en negrita, o permitir búsquedas de textos `importantes`.

Esta ventaja (que podemos resumir en la frase "un único marcado, múltiples usos") representa también el mayor inconveniente de SGML: determinar el tratamiento del documento. En efecto, si no tenemos definido una semántica concreta para las marcas, ¿cómo vamos a tratar los documentos que contengan dichas marcas para definir su estructura? Evidentemente se hace patente la necesidad de otros formalismos, más o menos relacionados con SGML, para poder realizar el tratamiento de los documentos SGML. Entre ellos podemos contar con lenguajes de programación de propósito general, los cuales deben manejar directamente (es decir, sin ningún API) los documentos, o el lenguaje DSSSL (*Document Style Semantic and Specification Language*) [ISO DSSSL] lenguaje diseñado para hacer transformaciones, y proporcionar un lenguaje de presentaciones para documentos SGML. En la práctica solo las grandes entidades, públicas o privadas, podían afrontar el coste de introducir SGML en su sistema informático, ya que al analizador SGML encargado de validar los documentos debía incorporarse el esfuerzo del tratamiento posterior de los documentos.

En lo referente al tercer punto, la supuesta independencia de la plataforma radica en no hacer ninguna interpretación del juego de caracteres del documento, mas allá de la de asignar una representación visual al código correspondiente. Por ejemplo, si reservamos el carácter ASCII de código 001 (cuya representación visual es:☺) para indicar texto en negrita en vez de la propia representación visual, estamos desvirtuando la aproximación SGML. Aunque el éxito de esta técnica está avalada por estándares como HTML, en el contexto SGML ligado en sus orígenes al mundo editorial presenta un grave inconveniente: restringir el juego de caracteres utilizado. SGML solventa este problema incluyendo en la *declaración SGML*, que hace explícito el juego de caracteres utilizado por el analizador, así como la interpretación que se hace de estos caracteres. XML extiende las posiblidades de SGML permitiendo la carga por parte del analizador del juego de caracteres según el cual está codificado el documento. La única solución válida, a nuestro entender, pasa por utilizar un juego de caracteres los suficientemente amplio, tal y como puede ser el juego de caracteres UNICODE.

Documentos SGML

Hemos dicho que el fin último de SGML es la de incluir un conjunto de marcas en un documento para poder capturar la estructura del mismo, y en base a esta poder especificar un tratamiento. Desde un punto de vista físico, toda esta información queda recogida en el *documento SGML*. Dicho documento está formado básicamente por tres partes consecutivas: la *declaración SGML*, la *declaración del tipo de documento*, y la *instancia del documento*.

La declaración SGML contiene información sobre las capacidades del analizador SGML tales como el juego de caracteres utilizado, las capacidades físicas (número de atributos, número de entidades declaradas, número de caracteres en las entidades, etc.), la sintaxis concreta utilizada así como otra serie de características. Al contener información inmutable (por lo general) sobre el analizador solía obviarse del documento físico SGML.

La declaración de tipo de documento (`DOCTYPE`), forma parte del *prólogo SGML* (declaración de tipo de documento, subdocumentos, y *Link Process Definition*), y es la forma de relacionar los datos marcados con la definición de la estructura a la cual deben ajustarse esos datos. Esa definición de estructura viene representada por la *Definición de Tipo de Documento*, o *DTD*. Es en la DTD donde se almacenan las producciones gramaticales que van a definir los datos, junto con sus etiquetas que vamos a considerar válidos.

La instancia del documento son los datos del lenguaje base a los cuales les hemos añadido las etiquetas del lenguaje de marcas con el fin de hacer patente esa estructura que no puede identificarse sin las marcas.

Por tanto, y en base a los anterior, si asimilamos SGML a lenguajes clásicos, ahora anteponemos las producciones (bien físicamente o bien por referencia) que definen al lenguaje C delante de cualquier programa C, con el fin de que el analizador valide si el programa se ajusta a la sintaxis de C. En este caso es necesario anteponer las reglas gramaticales porque en vez de producir un tratamiento concreto, nos interesa comprobar si la instancia del documento se ajusta a la estructura definida para cualquier estructura que nos interese definir. Es decir, en vez de *compilar* (como con los programas C) nos interesa *validar* sintácticamente, dejando la compilación o tratamiento para un paso posterior.

Declaraciones fundamentales en las DTDs

Las definiciones de tipos de documento o DTDs son las encargadas de definir que datos pueden aparecer en la instancia, y que etiquetas pueden aparecer dentro de estos datos para hacer explícita la estructura de los mismos. No tiene sentido en este contexto proporcionar una descripción exhaustiva de los componentes que pueden aparecer dentro de una DTD. Una referencia bastante valiosa para este fin (por supuesto aparte de [Goldfarb 90]) es [Maller 96].

El componente fundamental de las DTDs es aquel encargado de definir las etiquetas que pueden aparecer en la instancia del documento, así como el contenido de las mismas (relaciones jerárquicas con otras etiquetas o datos). Dicho componente es la *declaración de elemento*. En esta declaración aparece el nombre del elemento, la posibilidad de omitir la etiqueta de origen o de fin de elemento (por ejemplo en el elemento `dimspec` de la dtd arquitectónica de HyTime se pueden omitir ambas, ya que su contenido está formado por un origen y cantidad en un eje de coordenadas. Otro ejemplo es el elemento `HTML` de la DTD estricta de HTML 4.01) y el contenido del mismo. Este contenido puede ser vacío (`EMPTY`), estar formado por cualquier elemento declarado en la DTD (`ANY`), datos carácter sin referencia a entidades (`CDATA`), datos carácter con referencias a entidades (`RCDATA`), datos carácter con referencias a entidades o con otras etiquetas (`#PCDATA`; es decir contenidos mixtos de caracteres y datos), o un modelo de contenido, además de inclusiones o exclusiones. El contenido más interesante viene dado por el modelo de contenido, donde pueden aparecer caracteres u otros elementos relacionados mediante conectores de secuenciación (`,`), de opcionalidad (`|`), o de presencia en cualquier orden (`&`). A su vez los modelos de contenido pueden estar caracterizados por indicadores de ocurrencia de opcionalidad (`?`), repetición opcional (`*`), repetición obligatoria (`+`). De esta forma si queremos indicar que una descripción de un libro va estar formada por el libro y el autor del mismo, siendo opcional la etiqueta de cierra no tenemos más que proporcionar la siguiente declaración de elemento:

```
<!ELEMENT desc - O (libro, autor)>
```

Son estas declaraciones de elementos las que hacen las veces de producciones. Otro componente fundamental en las DTDs es la *declaración de atributo* que permite incluir información adicional dentro de las etiquetas. Aunque pudiéramos estar tentados a seguir con el paralelismo entre SGML y EBNF (atributos aparte) no vamos a proceder de dicha forma en este punto. La razón se debe a que las gramáticas atribuidas tienen una potencia expresiva muy superior a los atributos que pueden definirse en las DTDs SGML. De esta forma se puede utilizar una gramática atribuida con el fin de que unas producciones adquieran información (contextual) de otras producciones. Este extremo no es posible utilizando atributos SGML.

Las *declaraciones de entidades* SGML permiten definir texto de sustitución de manera interna (*entidad interna*) o externa (*entidad externa*) para ser utilizado en la DTD (*entidad parámetro*) o en la instancia del documento (*entidad general*). Para las entidades generales externas es posible definir una notación (`NOTATION`) encargada de procesarlas a petición de una llamada del procesador SGML (estos datos que el procesador no sabe como tratar se denominan *datos no SGML*). Es curioso como aparece aquí la idea de componente software pero bajo una perspectiva errónea. En principio estas notaciones se correspondían con programas ejecutables (es decir `.exe`) que eran llamados dinámicamente por el procesador. Una interpretación más amplia y productiva (al menos desde nuestro punto de vista) sería la de considerar a estas notaciones como componentes software que dinámicamente son incluidos por la aplicación que procesa el documento.

Otros componentes SGML

Dejamos para este apartado otros componentes susceptibles de aparecer tanto en la DTD como en la instancia del documento: las *secciones marcadas* y las *instrucciones de procesamiento*. Las primeras sirven para denotar partes del documento que deben ser tratadas o ignoradas por el analizador. Las segundas contienen información específica para las aplicaciones que manejan los datos SGML y deben ser obviadas ya que rompen el principio de independencia del tratamiento. Por supuesto, hay un caso distinguido en el que se debemos utilizar una instrucción de procesado, y es la *declaración SGML*. Por último no podemos terminar la descripción de SGML sin mencionar los comentarios, texto ignorado por el analizador incluido con el fin de proporcionar información al lector humano.

4.2.3 XML

El *eXtended Markup Language*, o XML [W3C XMLa], es una simplificación de SGML. Dicha simplificación se aplica a las capacidades expresivas del lenguaje, limitando las relaciones jerárquicas que pueden aparecer entre las marcas de los lenguajes definidos. El fin último de esta simplificación es la de construir analizadores más sencillos para su uso a través de Internet. Pues bien, aunque esta es la característica técnica más notable de XML en relación a SGML, en la práctica existe una diferencia mucho mayor: el enfoque de venta. Podemos decir que respecto a SGML, XML es el mismo perro con distinto collar, y ha sido el *World Wide Web Consortium* el encargado de cambiar el collar al perro de la *International Standard Organization*. Durante años, ISO fue la encargada de dirigir estandarizar y controlar el estándar SGML. Esto revirtió en serias restricciones para la difusión de dicho lenguaje, ya que incluso había que pagar a la organización para obtener un ejemplar del estándar (bueno, o comprar el libro de Goldfarb [Goldfarb 90]). Además dichos estándares estaban concebidos como textos de referencia, y no como textos de aprendizaje. También es cierto que la potencia de SGML hacía bastante difícil el desarrollo de aplicaciones para la gestión de documentos marcados, estando estas tecnologías al alcance únicamente de las grandes instituciones, públicas o privadas.

Al proporcionar el W3C el estándar XML (técnicamente "recomendación", ya que el W3C nunca proporciona estándares) se produjo un cambio de mentalidad de uso de los lenguajes de marcado, produciéndose una auge inusitado en el mundo de la informática. Es a partir de entonces cuando realmente se ha extendido el uso de las tecnologías de marcado a todos los ámbitos. Por lo demás XML no difiere en exceso de SGML. Veamos a continuación las diferencias más notables entre ambos estándares.

Variaciones significativas entre XML y SGML

Uno de los objetivos de diseño de XML fue: "3. XML será compatible con SGML" [W3C XMLb], por lo tanto y tal y como ya hemos anunciado las diferencias entre XML y SGML son mínimas. En efecto.

Quizás la mayor diferencia entre los documentos SGML y XML es que en los segundos la declaración de tipo de documento es opcional, ya que el uso para Internet de XML aconseja eliminar toda la información "superflua" que se pueda. Esto da lugar a distinguir entre *documentos bien formados* (aquellos que se ajustan a las normas sintácticas de XML) y *documentos válidos* (aquellos bien formados que se ajustan a una DTD concreta). Es importante darse cuenta que un documento bien formado del que no sepamos a que DTD se ajusta es tan útil como un bidón de combustible del que no sabemos si es gasolina normal o diesel. En otras palabras ¿cómo vamos a especificar un tratamiento basado en la sintaxis de un documento si no sabemos a que sintaxis se ajusta? Desde nuestro punto de vista, todo documento debe hacer referencia a su DTD (aunque sea a través de un comentario XML), ya que si no su utilidad es más bien limitada.

A pesar de que la anterior es la diferencia más llamativa, la más técnica es la limitación de los contenidos de los elementos. Ahora desaparecen los contenidos `CDATA` (salvo su inclusión explícita como *sección* `CDATA`) y `RCDATA`, el conector `&`, y los contenidos mixtos se limitan a aquellos de la `forma (#PCDATA|elem`$_1$`|...|elem`$_n$`)*`. Como podemos ver son simplificaciones dirigidas a construir analizadores más sencillos.

Otra variación que ya hemos comentando es que cada documento puede elegir el juego de caracteres en el que está codificado. Esto se consigue a través de la *declaración XML*, instrucción de procesado que además, y entre otras tareas, tiene por misión distinguir entre documentos SGML y documentos XML, evitando así problemas con aquellos documentos XML que no incluyan su DTD.

A pesar de no incluirse como parte constituyente del estándar, es posible utilizar sintaxis alternativas para definir las producciones gramaticales de XML [Lee 00]. Entre estas cabe destacar la recomendación W3C de *esquemas XML* [W3C schema], sintaxis alternativas a las DTDs que entre otras opciones permiten restringir la naturaleza de los contenidos de los elementos XML.

4.3 El lenguaje de estructuración hipermedia basado en tiempo (HyTime)

4.3.1 Introducción

Dedicamos un apartado especial de este capítulo al estándar HyTime (*Hypermedia/Time-based Structuring Language*) en su segunda edición [ISO HyTime]. HyTime no es una aplicación SGML ya que no es una DTD SGML. El estándar esta concebido en una doble vertiente. Por un lado proporciona una ampliación a las capacidades expresivas de SGML a través de la noción de *forma arquitectónica* (definida en las *Facilidades Extendidas* del estándar). Una forma arquitectónica, básicamente, es una manera asimilar un documento que se ajusta a una DTD con otro documento que se ajusta a otra DTD. La ventaja de esta aproximación es que la DTD destino (se supone que) tiene una semántica fija y determinada, pudiendo tratar el documento origen a través de la semántica de las marcas destino. Por otro lado, el propio estándar HyTime utiliza esta aproximación, siendo HyTime una DTD (técnicamente una DTD-arquitectónica) con semántica hipermedia. De esta forma se puede hacer el tratamiento hipermedia de un documento origen aprovechando la semántica de la DTD-arquitectónica de HyTime.

A pesar de la potencia expresiva de HyTime, este estándar nunca triunfó. A una primera edición, de cierto éxito como demuestra la existencia del libro de [deRose 94] ([Newcomb 91] también es una valiosa referencia), le siguió una segunda edición, la cual nunca llegó a prender en la comunidad hipermedia (ni siquiera hay un libro sobre este estándar). Una prueba de este fracaso la pueden proporcionar las páginas sobre HyTime de *Robin Cover* (*http://www.oasis-open.org/cover/)*, o la *Web de usuarios de HyTime* (*http://www.hytime.org)*, las cuales no han sido actualizadas en años. Las razones para este fracaso pueden ser dos. En primer lugar, el oscurantismo del estándar ISO. Como ya hemos comentado, los estándares ISO son un insuperable manual de referencia, pero son sumamente difíciles de ser utilizados como material didáctico. Como muestra sirva el dato de no incluir ejemplos (en la mayoría de los casos, y restringiéndonos a SGML, DSSSL, y HyTime). Esto añadido a una falta de bibliografía adecuada generó una incomprensión por parte de la comunidad hipermedia. En segundo lugar cabe citar la complejidad del estándar. En efecto, el manual de HyTime son varios cientos de páginas de material. Implementar una aplicación software que soportase todas las capacidades de HyTime conlleva un coste no asumible por la mayoría de organizaciones. En cualquier caso nos parece material de referencia obligada en el dominio hipermedia, y material por excelencia si nos restringimos al área de los lenguajes de marcado. Por otro lado, y debido a la dificultad de encontrar material sobre este estándar, el análisis de HyTime será ligeramente más extenso que el realizado sobre los demás estándares analizados en este capítulo.

4.3.2 Facilidades extendidas SGML

Las *Facilidades extendidas SGML* son un conjunto de extensiones a la funcionalidad de SGML. Aunque en teoría fueron concebidas para aplicaciones hipermedia y en la práctica no llegaron a utilizarse, presentan una extensión de SGML que puede ser utilizada en varios dominios, y no solo el hipermedia. A pesar de esto el más notable ejemplo de uso es la aplicación de dicha extensión en la definición del estándar HyTime. Están compuestas por seis cláusulas independientes entre sí:

- *Requisitos de definición de tipos léxicos*. La cláusula *Lexical Type Definition Requirements* (*LTDR*), contiene los requisitos para crear definiciones de los tipos léxicos utilizados en los conjuntos de propiedades, restricciones de tipos léxicos aplicadas a atributos de elementos y contenido, preguntas, y procesos de construcción de *data tokenizer* de *groves*. La idea de restringir los valores de los atributos de elemento, y de los contenidos de los mismos coinciden con las restricciones sobre los mismos objetos que se pueden imponer utilizando los esquemas XML.
- *Requisitos de definición de formas arquitectónicas*. La cláusula *Architectural Form Definition Requirements* (*AFDR*), determina la definición formal de *forma arquitectónica* mediante las cuales una *arquitectura de documento permisiva* gobierna la presentación SGML de sus documentos.
- *Requisitos de definición de conjuntos de propiedades*. La cláusula *Property Set Definition Requirements* (*PSDR*) contiene los requisitos para crear definiciones formales de los resultados de un proceso de análisis, conocido como *conjuntos de propiedades*. También describe los conceptos básicos de *groves*, conjuntos de estructuras de árbol construidas como resultado del análisis de documentos marcados. Es decir, esta cláusula contiene información sobre cómo construir el árbol de análisis de un documento y que información debería contener este.

- *Architectura general*. La cláusula *General Architecture* define las formas generales de la *Arquitectura General*, un conjunto de extensiones a la funcionalidad de análisis SGML.
- *Requisitos de definición formal de identificadores de sistemas*. La cláusula *Formal System Identifier Definition Requirements* (*FSIDR*) determina los requisitos para la definición formal de notaciones utilizadas en identificadores de sistemas.
- *El conjunto de propiedades SGML*. La cláusula *SGML Property Set* contiene el Documento de Definición de Conjunto de Propiedades SGML, incluidos los módulos opcionales para las facilidades extendidas SGML. Es decir, cómo se debe construir el árbol de análisis SGML y que información debe incluir este.

A continuación nos centraremos en las formas arquitectónicas y en los groves y conjuntos de propiedades, ya que representan la parte del estándar que necesitamos para comprobar como funciona HyTime (entendido como el lenguaje hipermedia).

4.3.2.1 Formas arquitectónicas

El estándar ISO define una *forma arquitectónica* como unas "reglas para la creación y procesado de componentes de un documento", identificando a la vez cuatro tipos posibles de formas: elementos, atributos, notaciones y atributos de notaciones. Desde un punto de vista SGML, una DTD arquitectónica (agrupación de formas arquitectónicas) no es más que una DTD tradicional SGML. La novedad radica en como validar el documento frente a la DTD arquitectónica. Ahora sobre el documento origen se impone una traducción de los elementos de la DTD origen a las formas arquitectónicas correspondientes. Otra característica nueva es la *naturaleza permisiva* de las arquitecturas, ya que los documentos que se ajustan a las DTDs arquitectónicas no tienen porque hacerlo en su totalidad, es decir, puede haber parte de los documentos que no tengan que validarse con la DTD arquitectónica. Nótese que esta es una nueva característica, pues hasta ahora las DTDs SGML eran de *naturaleza restrictiva*, ya que, ya que definían tipos de documentos completos, obligando a un ajuste total a la estructura definida.

En cuanto al procesado arquitectónico se distinguen dos pasos bien diferenciados. En un primer paso se valida el documento que contiene las marcas arquitectónicas frente a la DTD arquitectónica, y en un segundo paso una aplicación con una semántica concreta asignada a la DTD arquitectónica produce el tratamiento deseado del documento desde el punto de vista arquitectónico.

Antes de continuar con la teoría de arquitecturas, veamos un pequeño ejemplo que ilustre como funcionan. Supongamos que debemos procesar las direcciones postales de los clientes de varias empresas, las cuales tienen la información en formato SGML, pero con diferentes DTDs. Es en esta situación donde tiene sentido aplicar las arquitecturas, ya que pueden utilizarse como normas genéricas que se aplican a DTD concretas. La forma de definir una DTD arquitectónica es bastante similar a la forma de definir una DTD SGML (por no decir idéntica), con la única diferencia de utilizar letras minúsculas en las declaraciones. Por ejemplo una DTD arquitectónica que capture la dirección postal de cualquier persona puede ser:

```
<!-- DTD arquitectonica almacenada en dirPost.sgm -- >

<!element dirPostal - - (nombre,dir,cp,ciudad,pais) +(puente)>
<!attlist dirPostal urgente CDATA #IMPLIED
                    airMail CDATA #IMPLIED>
<!element (nombre | dir | cp | ciudad | pais) - O (#PCDATA)  >

<!element puente - O (#PCDATA|puente)>
```

Una DTD genérica de una empresa cualquiera puede ser:

```
<!ELEMENT postalAdd - - (name, add, zipc, city, state, country)>
<!ATTLIST postalAdd urgent (yes|no) no
                    airMail (yes|no) no>

<!ELEMENT name - O (fn, sn)>
<!ELEMENT (fn|sn) - O (#PCDATA)>

<!ELEMENT (add | zipc | city | state | country) - O (#PCDATA)>
```

Una dirección que se ajusta a la DTD anterior es:

```
<postalAdd urgent=yes airMail=yes>
    <name><fn>Antonio <sn>Navarro
    <add>Fac. Matemáticas Avda. Complutense s/n
    <zipc>28040
    <city>Madrid
    <state>Madrid</state>
    <country>ESPAÑA
</postalAdd>
```

Ahora debemos indicar que la información de la DTD de la compañía es posible entenderla como nuestra DTD arquitectónica. Aunque hay varias formas de hacer esto, una de ellas es indicarlo en la propia declaración del tipo de documento. En efecto:

```
<!DOCTYPE postalAdd [

<?IS10744 ArcBase dirPost>

<!NOTATION dirPost PUBLIC "-//Navarro//NOTATION Arquitectura Direccion Postal//SP">
<!ATTLIST #NOTATION dirPost
              ArcFormA NAME #FIXED dirPost
              ArcNamrA NAME #FIXED dpNames
              ArcDocF  NAME #FIXED dirPostal
ArcDTD   CDATA #FIXED dirPostDTD
              ArcBridF NAME #FIXED puente>

<!NOTATION   SGML   PUBLIC   "ISO   8879:1986//NOTATION   Standard   Generalized   Markup
Language//EN">
<!ENTITY dirPostDTD SYSTEM "dirPost.sgm" CDATA SGML>

<!ELEMENT postalAdd - - (name, add, zipc, city, country)>
<!ATTLIST postalAdd urgent (yes|no) no
                    airMail (yes|no) no
                    dirPost NAME #FIXED "dirPostal"
                    dpNames CDATA "urgente urgent">

<!ELEMENT name - O (fn, sn)>
<!ATTLIST name dirPost NAME #FIXED "nombre">

<!ELEMENT (fn|sn) - O (#PCDATA)>

<!ELEMENT (add | zipc | city | state | country) - O (#PCDATA)>
<!ATTLIST add dirPost NAME #FIXED "dir">
<!ATTLIST zipc dirPost NAME #FIXED "cp">
<!ATTLIST city dirPost NAME #FIXED "ciudad">
<!ATTLIST country dirPost NAME #FIXED "pais">
]>

<postalAdd urgent=yes airMail=yes>
    <name><fn>Antonio <sn>Navarro
    <add>Fac. Matemáticas Avda. Complutense s/n
    <zipc>28040
    <city>Madrid
    <state>Madrid
    <country>ESPAÑA
</postalAdd>
```

Una vez que pasa por el analizador, ahora el documento `postalAdd`, se ha convertido en el arquitectural `dirPostal`, en efecto:

```
<dirPostal urgente=yes airMail=yes>
    <nombre>Antonio Navarro
    <dir>Fac. Matemáticas Avda. Complutense s/n
    <cp>28040
    <ciudad>Madrid
    <pais>ESPAÑA
</dirPostal>
```

Nótese como no hemos tenido en cuenta a los elementos `fn`, `sn`, `state`, como hemos tenido que asimilar explícitamente el atributo `urgent` a `urgente`, y como el atributo `airMail` se ha asimilado automáticamente.

Debido a la naturaleza permisiva de las arquitecturas, es posible que un elemento de la DTD SGML se ajuste o no a una determinada arquitectura. Puede ser que un elemento de la DTD SGML:

- Se ajuste a un elemento de una arquitectura, en cuyo caso lo llamaremos *elemento arquitectónico*. También diremos que el elemento de la DTD SGML ha sido *asimilado* a una forma elemento.
- No se ajuste a ningún elemento de una arquitectura, en cuyo caso lo llamaremos *elemento no arquitectónico*.
- Se ajuste a varios elementos de diferentes arquitecturas.

Hasta ahora denominábamos DTD SGML a la DTD del documento original, otra posibilidad es referirse a esta como *DTD cliente* (o *DTD* simplemente), y a la DTD de la arquitectura que gobierna a la DTD cliente la denominaremos *DTD base*, o *DTD arquitectónica*. Respecto a esta cuestión debemos hacer notar que en diversos entornos a la DTD arquitectónica se la suele denominar *meta-DTD*, término que a nuestro parecer es no es el más adecuado, y que no usaremos a lo largo de este escrito. Al documento que contiene a la DTD cliente lo denominamos *documento cliente*, y definimos *documento arquitectónico* como al documento derivado del documento cliente, y que se ajusta a la DTD arquitectónica. En este documento arquitectónico solo aparecen elementos, atributos y notaciones que estén incluidas en la DTD arquitectónica.

A pesar de la naturaleza permisiva de las arquitecturas es posible que se produzcan errores en la validación arquitectónica si el documento resultante de aplicar el procesamiento arquitectónico no se ajusta a la estructura impuesta por la DTD arquitectónica. Por ejemplo si en el caso anterior no hubiésemos asimilado el elemento `country` con el elemento `pais` se hubiese producido un error arquitectónico. A pesar de esta restricción, no es necesario que el elemento raíz del documento arquitectónico tenga que coincidir con el elemento raíz de la DTD arquitectónica, pudiendo ser asimilado a algún otro elemento de la DTD arquitectónica.

También es posible una asimilación automática de la DTD cliente a la DTD arquitectónica basada fundamentalmente en la correspondencia de nombres entre los objetos de ambas DTDs (como en el caso del atributo `airMail`). Un elemento distinguido en las formas arquitectónicas es la *forma elemento puente* que sirve como un elemento genérico para mantener la estructura del documento origen en el documento arquitectónico. Suele caracterizarse por ser una inclusión en el modelo de contenido del elemento raíz de la forma arquitectónica (el elemento `puente` de la DTD arquitectónica anterior, el cual podíamos haber asimilado los elementos `fn` y `sn`).

4.3.2.2 Conjuntos de propiedades y *groves*

Como ya hemos comentado un *grove* no es más que el árbol de análisis de un documento SGML (o similar) cuyos nodos se encuentran interconectados entre sí. De hecho grove es el acrónimo de *Graph Representation Of property ValuEs*. Cada nodo de este árbol está formado por un conjunto de propiedades con sus valores correspondientes. Dichas propiedades pueden ser estructurales (definen la jerarquía del árbol de nodos), referenciales internas (referencias entre nodos del mismo grove), o referenciales no restringidas (referencias entre nodos de distintos groves, formando un *hipergrove*). La funcionalidad básica de los groves es permitir a diversos lenguajes como HyTime o DSSSL el poder referenciar las distintas partes que componen un documento, bien desde el punto de vista cualitativo (valores de atributos, por ejemplo), bien desde el punto de vista estructural (posición de los nodos dentro del grove).

Un *conjunto de propiedades* se encarga de definir los tipos de clases de nodos que pueden aparecer en un grove determinado. Por lo tanto a la hora de describir como debemos construir el árbol de análisis para un documento SGML utilizamos una descripción del conjunto de propiedades (a través de un documento SGML arquitectónico) que describe cuales van a ser las clases de nodos que vamos a tener, y que información deben contener dichos nodos (incluida la información estructural).

Los groves son *construidos* por *constructores de groves*, que no son más que programas que crean en memoria el grove de un documento según un conjunto de propiedades. Los conjuntos de propiedades y el proceso de construcción de groves definen el conjunto total de nodos que pueden aparecer en un grove. Sin embargo,

muchos conjuntos de propiedades definen clases de nodos y propiedades que no son usadas por todos los procesadores. Para cualquier contexto de procesamiento, podemos utilizar un *plan de grove* para especificar que clases de nodos y que propiedades son importantes en ese contexto. Esta información puede utilizarse para evitar gastar tiempo en construir información que no va a ser utilizada en un determinado contexto. Pueden utilizarse durante la construcción del grove para limitar la misma, o después de haber construido el grove, actuando como un filtro. Es importante darse cuenta que el plan del grove no indica al constructor de groves como construirlos, sino que le indica que cosas debe incluir y cuales excluir.

Todo conjunto de propiedades tiene asociado un plan de grove por defecto, que se utiliza si no se proporciona uno explícitamente. El plan de grove que incluye a todas las clases y propiedades de un grove es un *plan de grove completo* para ese conjunto de propiedades, y el grove construido según ese plan de grove es un *grove completo*. Un plan de grove se define nombrando las clases y propiedades que deben ser incluidas o excluidas del grove, y en función de la aplicación tenemos distintas formas de definir los planes de groves. Por ejemplo, HyTime define un conjunto de formas elementos para definir los planes de groves que serán usados en los documentos HyTime.

Además existen dos tipos de groves en función de su construcción: *groves primarios* y *groves auxiliares*. Los groves primarios son los construidos a partir de los datos fuente (por ejemplo un documento SGML). Los groves auxiliares son los construidos sobre otros groves. Ambos groves tienen diferentes reglas de definición y construcción, aunque coinciden en el resto de características. La principal diferencia es que los groves auxiliares deben ser capaces de referirse a los groves de los cuales fueron construidos. Por ejemplo, la arquitectura HyTime proporciona dos formas elemento de definición de groves, `pgrovdef` (*primary grove definition*) y `agrovdef` (*auxiliary grove definition*). Un elemento de definición de grove asocia datos fuente con el proceso de construcción de groves, además de permitir especificar el plan de grove.

Respecto a la construcción del grove hay dos cuestiones interesantes. La primera es que los groves están ordenados ya que las listas de nodos están ordenadas. Por lo tanto los nodos pueden ser localizados por su posición ordinal dentro de las listas de nodos que los contiene. Es el proceso de construcción de groves el que define el orden entre nodos. Por ejemplo para SGML el orden de los nodos del grove del documento esta determinado por el orden del documento fuente, bien en su declaración de orden o en el orden de la instancia (si no hay declaración explícita). Algunos tipos de datos puede ser que no necesiten orden. Una implicación del orden estricto es que el orden de los nodos en el grove está definido de forma determinista en términos de los datos fuente. Por tanto conociendo el orden de los datos fuente, y las reglas que determinan ese orden se puede obtener acceso a los nodos del grove simplemente por el orden de aparición. Esto es lo que hacemos con los grove de documentos SGML. La segunda es que conceptualmente los groves son estáticos, ya que una vez creado un grove no cambia. Este hecho favorece las transformaciones basadas en groves, ya que si estos cambiaran, dichas transformaciones no tendrían sentido. La Figura 4.1 ilustra el proceso de construcción de un grove.

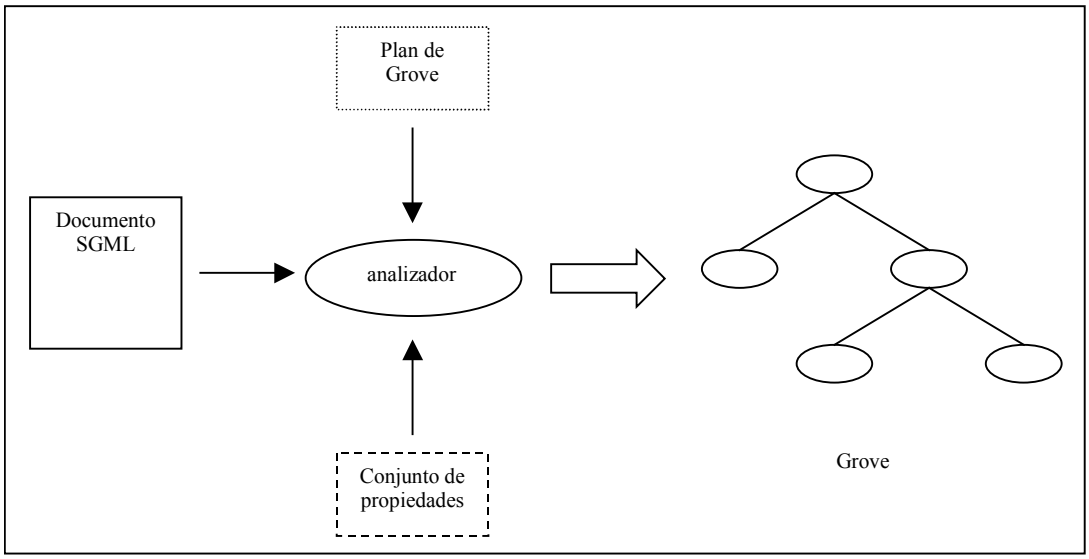

Figura 4.1 Construcción del grove de un documento

La Figura 4.2 (extraída de la página de Henry S. Thompson, *http://www.cogsci.ed.ac.uk/~ht/grove.HTML*) ilustra el grove que el analizador debería construir para el documento siguiente:

```
<!DOCTYPE simp [
  <!ELEMENT simp o o (bit*)>
  <!ELEMENT bit - - (#PCDATA)>
  <!ATTLIST bit name id #REQUIRED>
]>
  <bit name=one>1</bit>
  <bit name=two>2</bit>
```

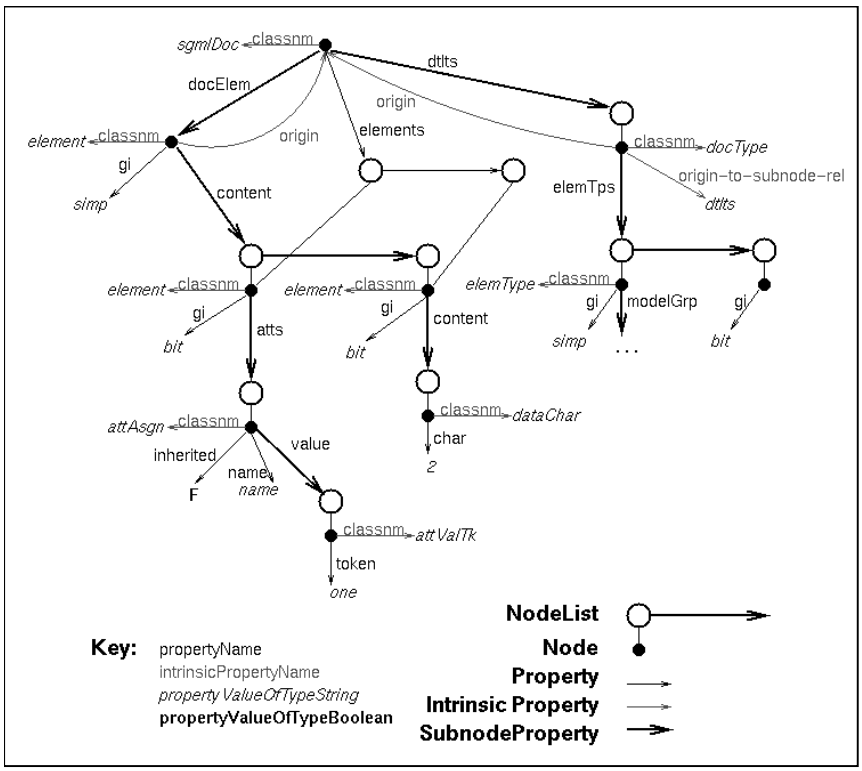

Figura 4.2 Un ejemplo de grove

Como podemos constatar, los groves son estructuras complejas. En el ejemplo, una DTD con dos elementos, y un documento con dos marcas genera una estructura como la mostrada en la figura 4.2.

En cuanto al procesamiento de groves, todo procesamiento SGML se inicia analizando el documento SGML. El resultado de este análisis es lo que llamamos *grove de análisis* o *pGrove* (*parse grove*), que además es un *grove primario*. Dicho pGrove es procesado para producir otro grove o una salida distinta de un grove. Cuando utilizamos arquitecturas podemos procesar un pGrove por un motor de arquitecturas para producir un grove que represente la instancia arquitectónica derivada del documento origen, el *grove arquitectónico* o *aGrove*. Los groves creados a partir de otros groves son los *groves derivados*. Los groves derivados que se crean para tareas específicas de procesamiento y no son independientes de los groves a partir de los cuales fueron creados se llaman *groves auxiliares*. Por tanto aGrove es un grove auxiliar. La Figura 4.3 resume la relación existente entre el pGrove y el aGrove.

Por último, cabe remarcarse que los groves son representaciones abstractas de datos. Los sistemas de procesamiento no necesitan construir literalmente el grove en sus estructuras de datos internos. Sin embargo, una representación literal de los groves es útil para validación, depuración, y, potencialmente intercambio entre distintas aplicaciones. El tipo de documento *representación canónica de grove* o *CGR* (*Canonical Grove Representation*) se utiliza para crear representaciones de groves basadas en caracteres, de tal forma que pueda haber una única representación canónica de groves para un grove o hipergrove dado. El tipo de documento CGR se define en el conjunto de Facilidades Extendidas de HyTime [ISO HyTime], y es un tipo de documento SGML derivado de la arquitectura HyTime, y aumentado por un conjunto de restricciones sobre la composición de los documentos fuente. Estas restricciones garantizan que las únicas diferencias posibles entre documentos CGR sean consecuencia de diferencias en los valores de las propiedades de los nodos.

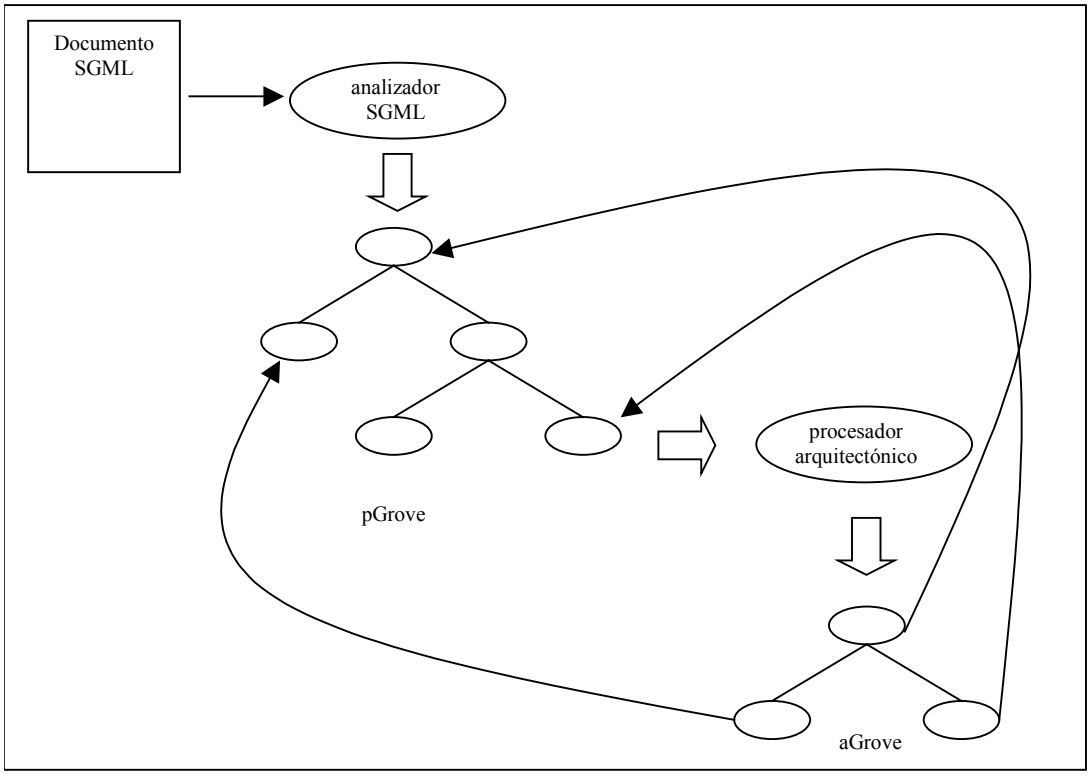

Figura 4.3 Relaciónes entre el aGrove y el pGrove

4.3.3 HyTime

4.3.3.1 Introducción

El estándar ISO dice "El lenguaje de estructuración hipermedia/basado en tiempo (HyTime), definido en este Estándar Internacional, proporciona facilidades para representar información dinámica y estática que es procesada e intercambiada por aplicaciones hipermedia y multimedia. HyTime es una aplicación ISO8879, el Standard Generalized Markup Language (SGML)". Es decir, básicamente es un método de representar aplicaciones hipermedia con el fin de servir como entrada de un sistema que la procesa y genera a la propia aplicación (de forma similar a como funciona HTML), o como formato de intercambio entre sistemas. En cuanto a ser aplicación ISO8879, lo es en la medida que el estándar original se amplíe con las Facilidades Extendidas definidas en el propio estándar HyTime. El estándar HyTime está formado por casi doscientas páginas (más casi otras trescientas de Facilidades extendidas y apéndices), representando un estándar difícil de ser aplicado por entidades de tamaño medio o pequeño.

HyTime está formado por cinco módulos:

- El *Módulo Base* está formado por diversas utilidades, algunas de ellas opcionales. Las obligatorias soportan la gestión de hiperdocumentos (utilizando SGML) y la identificación de propiedades de objetos. Las facilidades opcionales proporcionan tablas de acceso para elementos comúnmente utilizados, un mecanismo para asociar políticas de uso y acceso con objetos, y un mecanismo para relacionar atributos y el contenido de elementos a sus valores semánticos por referencia. El módulo base también define la notación de direccionamiento de coordenadas fundamentales utilizadas por el resto de módulos de HyTime.
- El *Módulo de Direcciones de Localización* permite la identificación de objetos que no pueden ser direccionados por los identificadores únicos SGML, y objetos que están en documentos externos. Existen tres tipos básicos de direccionamiento: localización por nombre, semántica, o por coordenadas.

- El *Módulo de Hiperenlaces* permite establecer relaciones entre objetos (hiperenlaces), bien dentro de un único documento o entre los objetos de información y documentos que constituyen un hiperdocumento.
- El *Módulo de Planificación* permite organizar eventos (ocurrencias de objetos) en ejes de coordenadas de espacios de coordenadas finitas, de tal forma que sus posiciones puedan ser expresadas en términos de sus relaciones entre ellos. La medida en los ejes de coordenadas puede hacerse en términos de unidades espaciales o temporales.
- El *Módulo de Presentación* puede utilizarse cuando esté activado el módulo de organización, y permite representar parámetros que gobiernen el proceso de presentación mediante modificación de objetos y/o proyección de eventos. La *Modificación de Objetos* permite especificar el orden en el que los objetos serán modificados durante la presentación, y también permite especificar los modificadores de objetos, como amplificadores y filtros que los afectarán (la semántica de los modificadores no esta definida por HyTime). La *Proyección de Eventos* permite proyectar eventos en espacios de coordenadas donde puedan ser percibidos, por ejemplo, de un espacio de coordenadas con un eje de tiempo virtual a uno basado en tiempo real. La proyección de eventos permite especificar los factores para calcular las posiciones y tamaños de los eventos en el espacio de coordenadas destino. En situaciones donde la posición de presentación y el tamaño de un evento no es predecible (por ejemplo si la interacción del usuario le afecta), las dimensiones virtuales de los eventos originales pueden ser proyectadas en espacio/tiempo real mediante lenguajes de expresiones definido por el usuario. Tales expresiones podrían soportar, entre otras cosas, valores de enlaces tardíos durante la presentación para resolver las posiciones y tamaños de los eventos proyectados (la semántica de los objetos de formateado para ajustarse a las nuevas medidas no está definida por HyTime).

La idea de funcionamiento HyTime es la siguiente. Los módulos anteriores se corresponden con distintos elementos de la DTD arquitectónica de HyTime. Utilizando una instancia de documento correspondiente al módulo de planificación se proporciona una descripción de la estructura espacio-temporal de la aplicación hipermedia (similar al esqueleto SMIL). Esta estructura puede ser un documento en si mismo, o un documento SGML sobre el que se ha impuesto la información arquitectónica. En el primer caso, evidentemente, el esqueleto espacio-temporal debe referir a sus contenidos. Entre dichos contenidos deben existir enlaces, los cuales son asimilados a enlaces HyTime definidos en el Módulo de Enlaces. El Módulo de direcciones de localización proporciona los mecanismos de direccionamiento utilizados fundamentalmente en el módulo de enlaces. Aunque no lo hemos mencionado hasta ahora, el módulo base fija las características fundamentales de HyTime, así como su estructura base. Finalmente el módulo de presentación sirve para describir como va a ser la presentación de los elementos descritos en un documento HyTime. Es decir, describe la semántica navegacional del documento HyTime. Este módulo es opcional, y en la práctica solamente tiene un propósito básicamente declarativo, ya que las aplicaciones que procesen los documentos HyTime deben tener esta semántica navegacional ya implementada. Podríamos decir que desde el punto de vista filosófico, la información que se puede proporcionar utilizando este módulo es similar a la información descrita por los conjuntos de propiedades: describen como debe funcionar una aplicación que procesa un documento SGML. En un caso para generar la presentación hipermedia, y en el otro para construir un grove.

El grado de aplicación de HyTime es variable en función del grado de compromiso declarativo deseado en la construcción de aplicaciones hipermedia. El caso expuesto en el párrafo anterior representa el caso máximo de declaratividad. En la práctica es posible que solamente se disponga de contenidos SGML enlazados según HyTime, y sea la aplicación la que teniendo codificada de alguna forma el esquema navegacional y la semántica de navegación la que los presente al usuario. En este caso obviaríamos tanto el uso del módulo de presentación como el de planificación.

4.3.3.2 Procesamiento basado en groves y HyTime

Ya sabemos que los groves se construyen por los constructores de groves según un plan de grove. Los constructores de groves son procesadores que toman como entrada el resultado de analizar datos o bien otro grove y producen un nuevo grove. Para HyTime, la entrada para un constructor de groves sería uno o más groves de documentos SGML y la salida sería un *grove semántico* HyTime tal y como se define en el conjunto de propiedades.

Para los motores HyTime, el grove semántico HyTime contiene los objetos definidos en el conjunto de propiedades. Los objetos en el grove semántico HyTime pueden estar derivados de distintos nodos de diferentes documentos. Un nodo evento, por ejemplo, podría está derivado de las formas elementos `event-`

`y extlist-` en un espacio de coordenadas finitas. La propiedad de extensión sería derivada de los elementos que forman la especificación de la extensión de los eventos. La Figura 4.4 muestra la construcción de un grove semántico HyTime de un pGrove y un aGrove.

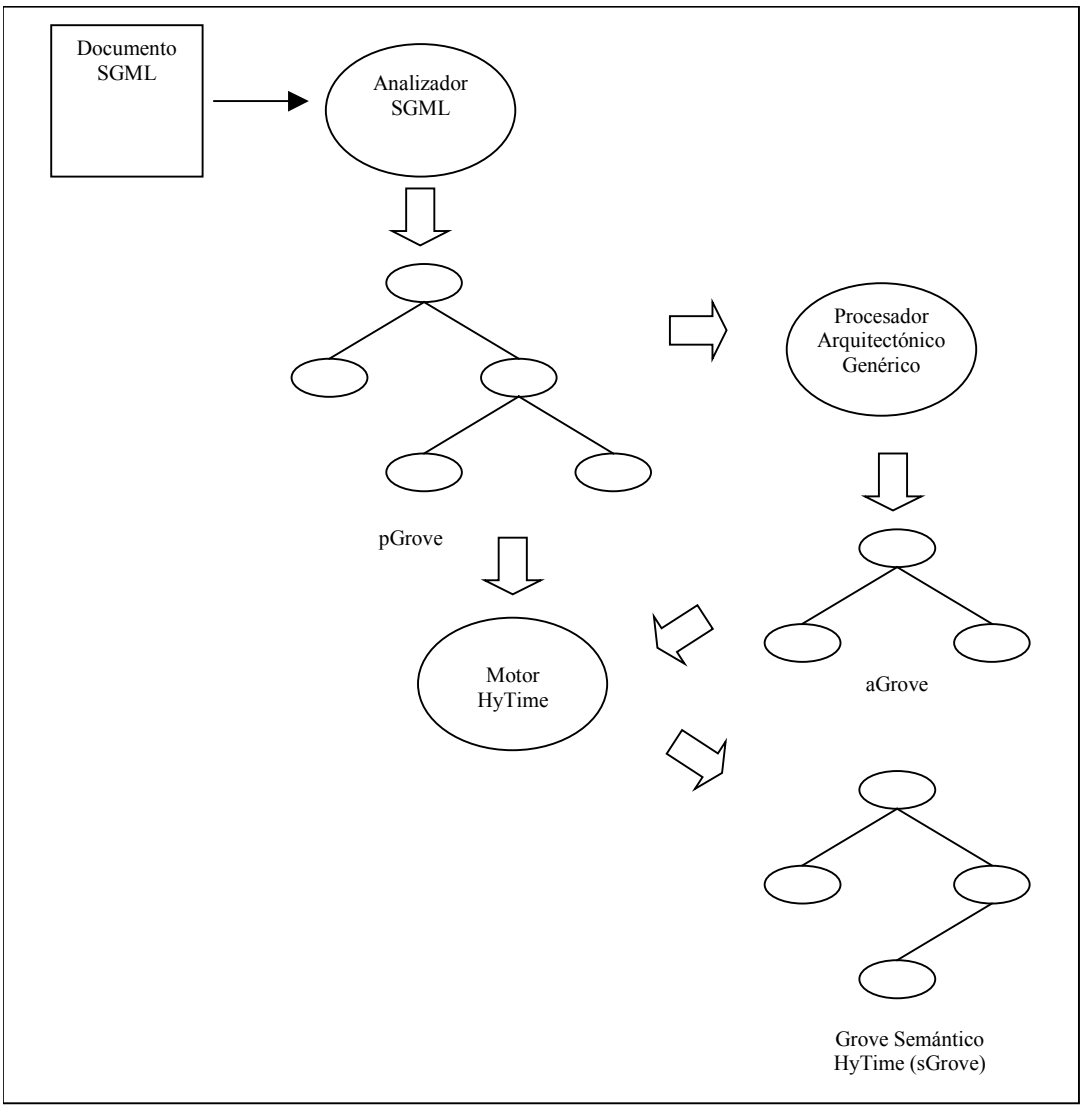

Figura 4.4 Construcción del grove semántico HyTime

Un motor HyTime utiliza el grove semántico junto con los otros groves en el hipergrove del cual el grove semántico es miembro, para hacer el procesamiento que sea necesario. Este procesamiento incluye la construcción de un nuevo grove para el documento original reflejando los resultados efectivos del procesado específico HyTime. Por ejemplo, si se utiliza la facilidad de localización basada en contenido, el motor HyTime debe resolver cualquier localización basada en contenido para determinar el contenido efectivo de estos elementos antes de que pueda resolver cualquier dirección de localización. Este nuevo grove el *pGrove efectivo*, o *epGrove* del documento cliente. Las Figura 4.5 muestra la construcción del epGrove a partir de los demás groves.

Una aplicación real probablemente no crearía una nueva representación en memoria del documento cliente, sino que simplemente aumentaría la representación existente. Sin embargo, es más sencillo hablar sobre el procesamiento abstracto si se representa como un proceso de creación independiente. Para mantener la abstracción grove más sencilla y hacer el direccionamiento de localizaciones más tratable, los groves se consideran como estáticos una vez creados. No hay noción de cambio sobre un grove una vez este ha sido creado. En particular, la posición de un nodo en un grove no puede cambiar una vez que ha sido asignado al grove. En el modelo de procesamiento abstracto el cambio se representa como la destrucción del viejo grove seguida por la creación de un nuevo grove. Las aplicaciones reales pueden tener, sin embargo, estructuras de datos reales más dinámicas.

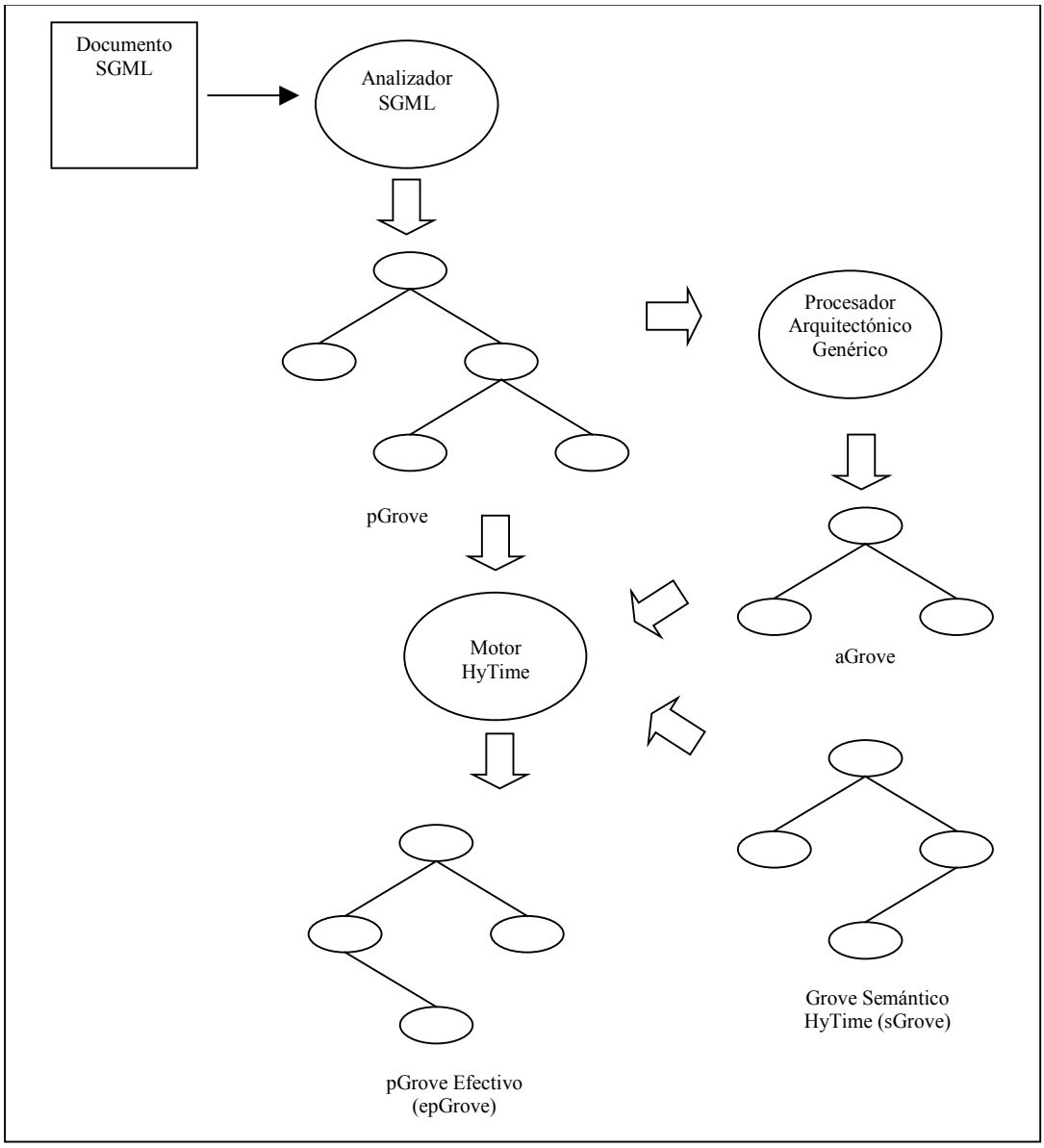

Figura 4.5 Construcción del pGrove efectivo (epGrove). Nótese que ahora el grove semántico se utiliza como entrada del motor Hytime.

Como podemos ver el procesamiento de documentos HyTime no es nada trivial. Veamos a continuación una breve descripción de los módulos HyTime.

4.3.3.3 El módulo base y el módulo de direcciones de localización

Módulo base

En este módulo se encuentra la información requerida para el uso del resto de módulos de HyTime. En particular la DTD arquitectónica del elemento raíz HyTime. Si desrefenciamos algunas entidades parámetro e importamos algunas arquitecturas de la arquitectura general, obtenemos que el elemento raíz HyTime tiene la siguiente forma:

```
<!element HyDoc – O (GABrid|HyBrid|fcs)* +(%link|%loc|%resorce)>
<!element GABrid – O (GABrid)*>
<!element HyBrid – O (GABrid|HyBrid|fcs)*>
<!element fcs (baton|evsched|wand)*>
<!entity % link "agglink|clink|hylink|ilink|varlink">
```

```
<!entity % loc "anchloc|bibloc|dataloc|fcsloc|linkloc|
                listloc|mixedloc|nameloc|nmsploc|pathloc|
                proploc|queryloc|relloc|treeloc">
<!entity %resorce "%resbase;|%resloc;|%resschd;|%resrend;">
```

Es decir, básicamente un documento HyTime (`HyDoc`) es una sucesión de elementos puente de la arquitectura general (`GABrid`), elementos puente de la arquitectura HyTime (`HyBrid`), o de espacios finitos de coordenadas (`fcs`). Además en cualquier punto de dicho documento puede aparecer un enlace HyTime (`%link;`), una dirección de localización (`%loc;`) o cualquiera del resto de formas arquitectónicas de HyTime (`%resource;`). Evidentemente aquí podemos distinguir varios tipos de documentos HyTime. El primer tipo define los esqueletos espacio-temporales de presentación, formados por una lista de eventos (`evsched`) incluida en un `fcs`. Otro tipo es similar al anterior pero define presentaciones (`baton`) o modificaciones de elementos en las presentaciones (`wand`). Otro tipo vendría dado por una sucesión de elementos puente HyTime (`HyBrid`) que capturan la estructura del documento base, apareciendo dentro de ellos (fundamentalmente) enlaces HyTime o especificaciones de localización.

Además en este módulo base se enumeran los direccionamientos disponibles en HyTime que serán implementados por el módulo de direcciones de localización: por nombre, por coordenadas o semántico.

Módulo de direcciones de localización

Este módulo permite localizar posiciones dentro de documentos SGML mediante los tres mecanismos anteriormente expuestos. Como ya hemos comentado, aunque en SGML el único mecanismo de localización de elementos es a través del atributo `ID`, en hipermedia puede ser necesario hacer localizaciones en partes de dichos elementos, con el fin de establecer las anclas de los enlaces. Además también es necesario referirse a elementos que no contienen un atributo `ID`.

Los tipos de objetos que pueden ser direccionados en HyTime, categorizados por la forma de direccionarlos son:

- *Nodos direccionados por un nombre* (localizaciones nombre-espacio)
  - *Entidades* (utilizando directamente referencias a entidades mediante atributos de tipo `ENTITY`, o utilizando direcciones de localización nombre-espacio, `nmsploc`).
  - *Elementos externos identificados*: elementos en otro documento con un atributo `ID` (utilizando direcciones de localización nombre-espacio, `nmsploc`).
  - *Elementos de documentos sin especificar*: el elemento raíz del documento a localizar no tiene atributo `ID` (utilizando direcciones de localización nombre-espacio, `nmsploc`).
    - *Elementos localmente identificados*: un elemento el documento origen que no es una dirección de localización pero tiene un atributo `ID` (por lo tanto no utiliza ningún otro mecanismo de direccionamiento).
    - *Valor de propiedad*: el valor de una propiedad de un nodo en un grove direccionado por un nombre de propiedad (utilizando la dirección de localización de propiedad, `proploc`).
- *Localizaciones direccionadas por una posición coordenada* (localizaciones coordenadas)
  - *Nodos de listas*: nodos en una lista de nodos direccionados por una posición dentro de la lista (utilizando la dirección de localización en lista, `listloc`).
  - *Nodos de árboles*: nodos en un árbol direccionados por sus posiciones dentro del árbol (utilizando la dirección de localización en árbol, `treeloc`), por sus posiciones dentro de una tabla de caminos de la raíz del árbol hasta las hojas (utilizando la dirección de localización de caminos, `pathloc`), o por sus relaciones genealógicas con otros nodos (utilizando la dirección de localización relativa, `relloc`).
  - *Objetos planificados*: objetos que aparecen dentro de una región de un espacio finito de coordenadas (`fcs`). Localizados bien por su localización coordenada (utilizando" direcciones de localización FCS, `fcsloc`), o por su especificación absoluta espacio-temporal (requiere la opción de la *especificación de calendario*). Requiere el módulo de planificación.
- *Objetos direccionados por una construcción semántica* (localización semántica)
  - *Nodos direccionados por preguntas sobre sus propiedades* (utilizando la dirección de localización por pregunta, `queryloc`).

- *Objetos inaccesibles por la computadora*: descripciones de un documento, persona, u otro objeto de información que no puede ser accedido automáticamente (utilizando la dirección de localización bibliográfica, `bibloc`).

Por ejemplo, una localización del estilo ID-IDREF SGML sería un ejemplo de nodo direccionado por nombre, en el caso de un elemento localmente identificado. Es decir,

```
... según los especificado por <autor biografia="Navarro">de este texto
</autor>...
```

es una referencia válida si el atributo `biografia` es de tipo `IDREF`, y hay un elemento cuyo valor de atributo `ID` es `Navarro`.

Por ejemplo, una localización direccionada por una posición coordenada en nodos de árboles sería:

```
<treeloc id="ejemplo" locsrc="arbol1">1 2 3</treeloc>
```

que localiza en el documento `arbol1`, el tercer hijo, del segundo hijo del nodo raíz.

Por ejemplo, una localización de un objeto direccionado por una construcción semántica de un objeto inaccesible por la computadora sería:

```
<bibloc>Todos los artículos de Antonio Navarro</bibloc>
```

Aunque este es uno de los módulos más potentes de HyTime, permitiendo incluso direccionamientos basados en direcionamientos, el material aquí expuesto puede servir para ilustrar sus principios básicos.

4.3.3.4 El módulo de hiperenlaces

Un *enlace* HyTime es una relación tipada entre dos o más objetos, donde cada uno tiene un papel en la relación. Como objeto abstracto, los enlaces HyTime tiene tres propiedades fundamentales:
- El tipo de enlace.
- Las anclas del enlace.
- Los roles semánticos que desempeñan las anclas en la relación representada por el hiperenlace (*los papeles de las anclas*, capturados por el atributo arquitectónico `anchrole`).

Además en principio no se supone ninguna direccionalidad en el enlace, siendo esta especificada por el atributo arquitectónico `linktrav`. Aunque hay catorce posibilidades para este atributo. Cuando veamos al enlace `hylink` comentaremos las posibilidades más utilizadas.

HyTime define cinco formas elemento para representar enlaces:
- *Hiperenlace*, `hylink`. Es el más flexible, pudiendo tener varias anclas independientes del propio enlace.
- *Enlace contextual*, `clink`. Representa enlaces contextuales unidireccionales.
- *Enlace agregado*, `agglink`. Representa relaciones de agregación.
- *Enlace independiente*, `ilink`. Proporciona una sintaxis alternativa a los `hylink`, en la cual las direcciones de las anclas se especifican con atributos IDREF, en vez de con atributos separados para cada papel de ancla.
- *Enlace variable*; `varlink`. Representa hiperenlaces en los cuales los papeles de las anclas pueden variar entre instancias del mismo tipo de enlace.

En la práctica los enlaces `hylink`, `ilink`, y `varlink` son funcionalmente equivalentes pudiendo sustituirse unos por otros sin perdida de poder expresivo. Además las formas `clink` y `agglink` no son más que formas especializadas de la forma `hylink`. Respecto al tratamiento, la mayoría de veces los enlaces se encuentran en un documento externo a aquel que contiene las anclas (excepto para enlaces `clink`), siendo este procesado para poder navegar los enlaces de los documentos enlazados.

hylink

Representa el recurso fundamental de hiperenlaces en HyTime. Supongamos que queremos representar la relación "es capital de" que se establece entre un país y una ciudad. Para representar esta relación a través de un enlace `hylink` no tenemos más que considerar el siguiente elemento:

```
<!ELEMENT esCapitalDe - O EMPTY>
<!ATTLIST esCapitalDe
       HyTime NAME #FIXED "hylink"
       linktrav NAMES #FIXED "E D"
       anchrole CDATA #FIXED "pais capital"
       pais IDREF #REQUIRED -- dirección del ancla pais -->
       capital IDREF #REQUIRED  -- dirección del ancla capital --
```

Por ejemplo, ahora podemos tener:

```
<esCapitalDe capital=Madrid pais=España>
...........
<paises>
   <pais id=España>........</pais>
.....
<ciudades>
   <ciudad id=Madrid>........</ciudad>
```

De esta forma indicamos que el elemento `esCapitalDe` se va a asimilar a un elemento `hylink` (`HyTime NAME #FIXED "hylink"`), que va a tener dos anclas llamadas `pais` y `capital` (`anchrole CDATA #FIXED "pais capital"`), y que el recorrido va a ser unidireccional del `pais` a la `capital` (`linktrav NAMES #FIXED "E D"`). En particular de los catorce tipos de especificaciones `linktrav` las más corrientes son:

- `A A`, para enlaces bidireccionales.
- `E D`, para enlaces unidireccionales sin vuelta atrás.
- `E RD`, para enlaces unidireccionales con vuelta atrás.

El estándar también permite representar enlaces n-arios, sin más que especificar una lista de `IDREFS` como una de las anclas.

clink

Representa a los enlaces contextuales unidireccionales, es decir, aquellos en los que el elemento que constituye el enlace también contiene al ancla origen (como en HTML). Así por ejemplo, podemos tener:

```
<!ELEMENT a - - (#PCDATA)>
<!ATTLIST a
       href IDREF #REQUIRED
       HyTime NAME #FIXED "clink"
       HyNames CDATA #FIXED "linkend href">
```

De esta forma indicamos que el elemento a va a ser un enlace contextual (`HyTime NAME #FIXED "clink"`), y que el atributo que referencia al destino va a ser `href` (`HyNames CDATA #FIXED "linkend href"`).

agglink

Representan relaciones similares a las establecidas entre padres e hijos de elementos, pero establecidos entre elementos no ligados de esta forma. Así por ejemplo podemos tener:

```
<!ELEMENT ciudadesDe - O EMPTY>
<!ATTLIST ciudadesDe
       HyTime NAME #FIXED "agglink"
       anchrole CDATA #FIXED "pais ciudades #LIST"
       pais IDREF #REQUIRED
       ciudades IDREFS #REQUIRED>
```

De esta forma indicamos que `ciudadesDe` es un `agglink` (`HyTime NAME #FIXED "agglink"`), que el origen está en `pais` y que los agregados son `ciudades` (`anchrole CDATA #FIXED "pais ciudades #LIST"`).

ilink

Es como hylink, pero utiliza un único atributo (`linkends`) para direccionar sus anclas, en vez de un atributo para cada ancla. Por ejemplo

```
<!ELEMENT esCapitalDe - O EMPTY>
<!ATTLIST esCapitalDe
        HyTime NAME #FIXED "hylink"
        anchrole CDATA #FIXED "pais capital"
        linkends IDREFS #REQUIRED>
```

Ahora la información se codifica de la siguiente manera:

```
<esCapitalDe linkends="España Madrid">
...........
<paises>
   <pais id=España>........</pais>
.....
<ciudades>
   <ciudad id=Madrid>........</ciudad>
```

varlink

Esta forma difiere de `hylink` en dos puntos: utiliza subelementos, en vez de atributos, para direccionar sus anclas, y no es necesario que los papeles de las anclas sean constantes para un tipo de enlace determinado. De esta forma:

```
<!ELEMENT enlace  - - (ancla+)>
<!ATTLIST enlace
        tipo NAME #IMPLIED
        HyTime NAME #FIXED "varlink">

<!ELEMENT ancla O O (#PCDATA)>
<!ATTLIST ancla
        anchrole NAME #IMPLIED
        linktrav NAME "A"
        listtrav NAME "N"
        loctype CDATA #FIXED "#CONTENT IDLOC"
        HyTime NAME #FIXED "anchspec">
```

De esta forma definimos un enlace, cuyas anclas son elementos que lo componen (tal y como exige el estándar), siendo los valores de las anclas el contenido `#PCDATA` que declaran.

Nota

Aunque en los ejemplos nos hemos limitado a utilizar únicamente IDREF como métodos de direccionamiento, la verdadera potencia de HyTime consiste en utilizar además las capacidades expresivas proporcionadas por el Módulo de direcciones de localización, indicando anclas fuera del documento que contiene el enlace (salvo para enlaces `clink`).

4.3.3.5 Los módulos de planificación y presentación: definiendo aplicaciones hipermedia

El *módulo de planificación* va a ser el encargado de definir los esquemas navegacionales y/o presentacionales que se van a aplicar a una serie de contenidos hipermedia. Para ello (básicamente) define un conjunto finito de coordenadas espaciales, temporales o espacio-temporales (`fcs`), sobre los que aparecen los componentes audiovisuales de la presentación (los marcos de la presentación, por ejemplo). A esos componentes que aparecen dentro de un `fcs` se les denomina eventos (`event`; a pesar de su nombre no tienen nada que ver con los eventos tradicionales en programación), y aparecen dentro de una lista de eventos (`evtsched`). Los eventos que aparecen dentro de un `fcs` deben definir (básicamente) su extensión (`exspec`) y el contenido que muestran, bien explícitamente o por referencia (`conloc`). En ambos casos puede hacerse por contenido directo o por referencia. Así, por ejemplo si queremos considerar una ventana formada por dos marcos, que contienen un índice y unos contenidos, no tenemos más que escribir:

```
<ventana HyTime="fcs">
  <listaMarcos HyTime="evtsched">
   <marco HyTime="event"
          exspec="dimensionesMarco1"
          conloc="indice.sgm">
   <marco HyTime="event"
          exspec="dimensionesMarco2">
          conloc="contenidos.sgm">
  </listaMarcos>
</ventana>
```

El documento anterior es bastante sencillo, pero tiene un serio problema: en ningún sitio se especifica que los contenidos que aparecen en el segundo marco tengan que hacerlo según la selección de enlaces del índice que aparece en el primer marco. Aunque, por supuesto, es posible proporcionar especificaciones de este comportamiento, estas están ya fuera del estándar HyTime.

Por otro lado, el *módulo de presentación* define una serie de formas que permiten representar las relaciones de representación entre eventos origen y eventos concretos de presentación. Aunque factible, este no es el método más razonable de especificar presentaciones, siendo más realista la utilización de aplicaciones que directamente implementan la semántica de presentación a partir de `fcss`, o la traducción a otros formatos con semántica definida.

Nótese que a pesar de la potencia expresiva de HyTime no hemos conseguido expresar modificaciones en la presentación (`fcs`) en base al evento mínimo que se espera en un sistema hipermedia: la selección de hiperenlaces. Estándares menos ambiciosos, como HTML si que son capaces de especificar este comportamiento, comprometiendo eso sí, la independencia de los contenidos frente al esquema navegacional. En el lenguaje propuesto en esta tesis (Capítulo 6) si que será posible especificar este tipo de comportamientos, por supuesto con independencia de contenidos/esquema navegacional, gracias a la semántica de navegación por defecto proporcionada por el modelo hipermedia definido en el Capítulo 5, Pipe.

4.4 Aplicaciones de SGML y XML

En este apartado veremos un conjunto de lenguajes de marcado descritos mediante SGML (HTML) y/o XML (todos). De la ingente cantidad de aplicaciones disponibles hemos elegido basándonos en un doble criterio: lenguajes de marcas relacionados con el dominio hipermedia, y lenguajes de marcas relacionados con descripción de interfaces de usuario. El interés en los primeros está claro, ya que pueden verse como concreciones prácticas de modelos hipermedia. Por ejemplo, las aplicaciones descritas mediante HTML quedan modeladas por el modelo HAM. Igualmente los documentos SMIL se ajustan a las estructuras descritas por el modelo Amsterdam. Respecto a los segundos, consideramos su inclusión en este apartado ya que en el Capítulo 6 se propondrá el uso de mecanismos similares para estructurar esquemas navegacionales. Aunque en esta tesis se proporciona un mecanismo propio basado en lenguajes de marcado nos parece interesante compararlo con otras aproximaciones existentes.

4.4.1 El lenguaje de marcado de hipertexto (HTML)

Introducción

Sin duda alguna el lenguaje HTML [W3C HTML], estándar W3C, es el representante más exitoso de los esfuerzos desarrollados en el dominio hipermedia (en [Berners-Lee 01] puede encontrarse una completa evolución de este lenguaje). Desde el punto de vista de este dominio, solamente tiene enlaces unarios, no permite incluir información sobre sincronización, no separa los contenidos de la navegación (pensemos en los *frames*), no describe aplicaciones ejecutables, sino que necesita de un interprete (el navegador), y por último un documento HTML puede contener más errores que aciertos pero el navegador, en la mayoría de los casos, conseguirá mostrar alguna información al usuario. Además, también representa el lenguaje de marcado de mayor éxito hasta el momento y eso que la mayoría de la gente que utiliza HTML es incapaz de leer una DTD.

Es decir, HTML es un lenguaje de marcado para describir aplicaciones hipermedia utilizado por miles de personas que no saben lo que es una aplicación hipermedia ni un lenguaje de marcado. Entonces ¿cuál es la razón del éxito de este lenguaje que permite aplicar conceptos desconocidos por la gente para construir un tipo de aplicaciones que ni siquiera saben que existen? Pues sin duda alguna su sencillez. HTML aprovecha al máximo la legibilidad que proporcionan los lenguajes de marcado para que los usuarios editen directamente las páginas Web. Es más, si los usuarios no quieren editar directamente las páginas pueden utilizar un editor *WYSIWYG* (Microsoft FrontPage, por ejemplo) para construirlas. El desarrollo de tales editores no supera en complejidad al de los editores de texto actuales, poniendo de manifiesto que la sencillez de HTML vuelve a ser una de sus principales ventajas.

En base a los párrafos anteriores parece que podríamos concluir que HTML es la solución idónea a todos nuestros problemas de desarrollo hipermedia, pero en la práctica no es así, ya que su mayor ventaja a la vez es su mayor inconveniente. Tal y como exponen [Sperberg-McQueen 94] y [Burnard 97] HTML no proporcionan la estructuración necesaria para hacer un tratamiento de los documentos HTML más allá del meramente presentacional. Además, las capacidades hipermedia proporcionadas por HTML son bastante limitadas, tal y como hemos expuesto en el primer párrafo. Estas son las razones por las cuales han surgido estándares como CSS para paliar las deficiencias de HTML, u otros como HyTime, o SMILE, expresamente concebidos como lenguajes para dar verdadero soporte a las aplicaciones hipermedia o multimedia, respectivamente.

A pesar de estos nuevos estándares, HTML continúa siendo referencia obligada en el contexto hipermedia. De hecho, este lenguaje forma parte de las tecnologías utilizadas en el Capítulo 6 de esta tesis como parte de un generador automático de prototipos, el cual utiliza transformaciones XSLT para convertir datos XML en páginas HTML que son cargadas en `JEditorPane`s del API Swing de Java.

Veamos a continuación las principales características de XHTML [W3C XHTML] en su versión 1.0, que no es más que una reformulación de HTML 4.0 en XML (y por supuesto con los cambios introducidos en HTML 4.01). Por otro lado no consideramos comportamientos dinámicos utilizando lenguajes de programación (de *script* o no) invocados directamente desde páginas HMTL.

Descripción técnica

Desde el punto de vista de una caracterización hipermedia las aplicaciones originalmente descritas por el lenguaje HTML podían ser modeladas a través de la máquina abstracta de hipertexto. Es decir, las primeras aplicaciones HTML estaban formadas por una estructura de grafo, cuyos nodos son las páginas HTML y cuyos arcos son los enlaces descritos por las anclas insertadas en dichas páginas. En efecto, si estudiamos la DTD de XHTML (Apéndice A del estándar) vemos como las páginas HTML en la práctica están formadas por una amalgama de tablas, listas, párrafos, etc., dentro de los cuales pueden aparecer anclas de enlaces HTML. Aunque técnicamente la imposición de esquemas navegacionales más complejos (es decir, *frames*) era práctica común en los navegadores previos a 1997, no se formalizó este hecho hasta la versión 4.0 de diciembre del 97. Esta característica invalida la norma de visualizar un único nodo activo del grafo, ya que ahora es posible tener activos dos o más nodos del grafo (páginas HTML) simultáneamente. El uso de *frames* hace necesario la aplicación de modelos más potentes (como Labyrinth, por ejemplo), para poder caracterizar a este tipo de aplicaciones. Estudiemos por tanto los elementos HTML de nuestro interés: las anclas y los marcos (o *frames).*

Las anclas HTML vienen definidas por el elemento `a`, en el cual entre otras se puede incluir información sobre si es un ancla origen (atributo `href`) o si es un ancla destino (atributo `name`). En la práctica esto determina enlaces unarios unidireccionales, por otra parte los más extendidos en el campo hipermedia. Nótese que en un enlace debe constar el contenido origen, el ancla origen, el contenido destino y el ancla destino. ¿Dónde codifica esta información HTML? Pues básicamente en contenido origen. En efecto, el contenido origen da soporte al ancla origen y esta codifica tanto el contenido destino como el ancla destino. En efecto consideremos la página HTML:

```
<html>
 <head>
  <title>Ejemplo</title>
 </head>1
 <body>
  <p>
```

```
   <a href="http://www.ucm.es/ucmd/centros.htm#FACULTADES">Facultades de
                                                          la UCM</a>
  </p>
 </body>
</html>
```

En este contenido origen podemos ver el ancla origen (`<a href="...">... </a>`), el contenido destino (`http://www.ucm.es/ucmd/centros.htm`), y el ancla destino (`#FACULTADES`). Es decir, en HTML la función de **resolución** del modelo Dexter es la función identidad (la propia especificación del destino es un identificador), y la función de **acceso** viene implementada por el propio navegador HTML (que a su vez implementa la semántica de HTML), el cual devuelve la página indicada por el valor del atributo `href`, en la posición indicada por el ancla destino allí contenido (de no existir dicha ancla, muestra el principio del documento).

Hasta este momento, y debido a que el esquema navegacional que se puede imponer sobre las páginas HTML se corresponde con el que permite el modelo HAM, HTML mantiene una separación entre el nivel de almacenamiento y el de presentación. Dicho de otra forma, el esquema navegacional que se puede imponer sobre los contenidos es independiente de estos. Al incluir los marcos esta norma se rompe, ya que las anclas de los enlaces entre contenidos deben ser las responsables de indicar en que marco deben mostrarse los contenidos destino del enlace. Esto provoca un acoplamiento inaceptable (desde el punto de vista de la reutilización) entre los contenidos y sus relaciones y el esquema navegacional que se quiere imponer sobre ellos. Dicho de otra forma, pervierte la separación entre el nivel de almacenamiento y el de presentación del modelo Dexter. Veamos en un sencillo ejemplo donde está el problema. Supongamos que queremos indicar la presencia de una ventana dividida en dos marcos. En uno de ellos se muestra un índice, y en el otro contenidos accesibles desde ese índice:

```
<html>
  <head>
    <title>Ejemplos 6</title>
  </head>
  <frameset cols="*, 3*" >
    <frame src="indice.htm"/>
    <frame name="marcoD"/>
  </frameset>
</html>
```

Si queremos indicar que los contenidos enlazados desde `indice.htm` aparezcan en el marco `marcoD`, debemos hacerlo explícito en las anclas del `indice.htm`. Es decir,

```
<html>
  <head>
    <title>Índice</title>
  </head>
  <body>
    <ul>
     <li><a href="cont1.htm" TARGET="marcoD">Contenido 1</a></li>
     <li><a href="cont2.htm" TARGET="marcoD">Contenido 2</a></li>
     <li><a href="cont3.htm" TARGET="marcoD">Contenido 3</a></li>
     <li><a href="cont4.htm" TARGET="marcoD">Contenido 4</a></li>
     <li><a href="cont5.htm" TARGET="marcoD">Contenido 5</a></li>
  </body>
</html>
```

Esto invalida reutilizar el mismo índice en distintos contextos, ya que liga sin solución de continuidad los enlaces entre el índice y los contenidos con su presentación en el marco `marcoD`.

Además esta aproximación HTML también invalida la posibilidad de que una página formada por marcos pueda variar dinámicamente los contenidos asociados por defecto a cada marco cuando se carga por primera vez la página de los marcos (aquella que contiene al elemento `frameset`). De esta forma si queremos tener una ventana formada por varios marcos en los cuales siempre aparece la misma información por defecto (música, imágenes, etc.) salvo en un único marco (pongamos que los cinco contenidos del índice anterior ahora queremos que aparezcan en una ventana con varios marcos, siendo uno de ellos el reservado para los contenidos), debemos crear tantas páginas con marcos como contenidos distintos queramos cargar en el único marco que varía (es decir, cinco páginas `frameset`, y en cada una de ellas se mantiene la página por

defecto asignada a cada marco, salvo la de aquel marco que debe cargar por defecto cada contenido seleccionado). El problema radica en que en este caso el índice deber referenciar a páginas con marco, y no a marcos concretos de páginas, es decir debería ser de la forma:

```
<html>
  <head>
    <title>Índice</title>
  </head>
  <body>
    <ul>
     <li><a href="marcoCont1.htm">Contenido 1</a></li>
     <li><a href="marcoCont2.htm">Contenido 2</a></li>
     <li><a href="marcoCont3.htm">Contenido 3</a></li>
     <li><a href="marcoCont4.htm">Contenido 4</a></li>
     <li><a href="marcoCont5.htm">Contenido 5</a></li>
   </ul>
  </body>
</html>
```

Donde cada `marcoCont_` mantiene la misma información, salvo la correspondiente a cada `Contenido i`. Aunque es posible resolver este problema utilizando, por ejemplo, JavaScript, esto invalida la ausencia de código de los documentos HTML.

En la aproximación presentada para el prototipado de aplicaciones en el Capítulo 6 de esta tesis, si que será posible el separar la información de enlaces de la navegacional, y asignar a una misma pantalla distintos contenidos que varían en función del enlace seleccionado y que se cargan por primera vez al inicializar la pantalla. Por supuesto, todo ello utilizando exclusivamente lenguaje de marcado (en el uso estándar promovido por esta tesis), y pudiendo ser visionado sin necesidad de un interprete.

4.4.2 El lenguaje de integración de multimedia sincronizada (SMIL)

Introducción

El lenguaje SMIL, *Synchronized Multimedia Integration Language*, [W3C SMIL] es un estándar W3C (llegó a recomendación en su versión 2.0 el 07 de agosto del 2001) utilizado para describir presentaciones multimedia, al igual que HTML se utiliza para describir aplicaciones hipermedia. Uno de sus objetivos de diseño es: "Definir un lenguaje basado en XML que permita a los autores escribir presentaciones multimedia. Utilizando SMIL 2.0, un autor puede describir el comportamiento temporal de una presentación multimedia, asociar hiperenlaces con objetos de distinta naturaleza y describir la distribución de la presentación en la pantalla".

La versión 2.0 es bastante extensa (el documento del estándar excede las seiscientas páginas de información), repercutiendo esta potencia en un alto nivel expresivo. Básicamente SMIL implementa las ideas fundamentales del modelo Amsterdam, proporcionando un lenguaje concreto a las construcciones del entorno CMIFed (en la sección 3.2.2 del Capítulo 3 vimos las relaciones existentes entre este entornos y las herramientas de desarrollo SMILE). Aunque Amsterdam es un verdadero modelo hipermedia, en la práctica, la implementación de SMIL se centra más en las capacidades multimedia que en las hipermedia. El estándar está basado en varios módulos agrupados de la siguiente manera.

- *Módulos de animación*. Son dos módulos que permiten describir cómo incorporar animación en una determinada línea temporal.
- *Módulos de control de contenido*. Son cuatro módulos que permiten elegir el contenido a mostrar de manera dinámica, así como la manera de obtener dichos contenidos.
- *Módulos de diseño*. Son cuatro módulos que permiten describir la distribución física de los contenidos en una presentación.
- *Módulos de enlace*. Son tres módulos que permiten describir el establecimiento de enlaces entre los distintos objetos que componen una presentación SMIL.
- *Módulos de objetos*. Son siete módulos que permiten describir los objetos de diverso tipo (audio, vídeo, etc.) que pueden aparecen en una presentación SMIL.
- *Módulo de metainformación*. Este módulo proporciona los mecanismos básicos para estructurar un presentación SMIL.

- *Módulo de estructura*. Este módulo permite estructurar los documentos SMIL.
- *Módulo temporal y de sincronización*. Este módulo permite describir la coordinación y sincronización de la presentación de diversos objetos a lo largo del tiempo.
- *Módulo de manipulaciones temporales*. Este módulo permite especificar manipulaciones avanzadas del comportamiento temporal, tales como la velocidad de reproducción de un elemento.
- *Módulo de efectos de transición*. Este módulo permite especificar las distintas transiciones audiovisuales que pueden existir entre objetos del mismo medio. Por ejemplo, si queremos sustituir una imagen por otra, es posible que la primera desaparezca por completo y sea sustituida por la segunda, o que la primera vaya siendo por partes (por ejemplo de izquierda a derecha) por la segunda.

Un estudio en profundidad de este estándar bien podría constituir un libro considerable tamaño, y además queda fuera de los objetivos de este trabajo. Por lo tanto, nos centraremos en los módulos de mayor interés para nuestro trabajo: el de estructura, sincronización, diseño y enlaces.

Descripción técnica

El *Módulo de Estructura* es el encargado de estructurar los documentos SMIL. En la práctica estos documentos son equivalentes a descripciones concretas de componentes compuestos del modelo Amsterdam. El elemento raíz del todo documento SMIL se denomina `smil`, siendo su contenido variable en función de los módulos que se incorporen en el lenguaje. El contenido de dicho elemento se subdivide en los elementos `head` y `body`, al igual que en HTML. El elemento head contiene (i) metainformación acerca del documento; (ii) información sobre la distribución física de los elementos que componen la presentación; y (iii) contenido definido por el usuario. Este contenido permite definir a los autores sus propios atributos de prueba para ser usados con el elemento `switch` perteneciente al Módulo de Control de Contenido Básico. Al igual que con el elemento `smil`, el elemento `body` presenta un contenido variable en base a los módulos incorporados en el lenguaje. Este elemento contiene información relacionada con el comportamiento temporal y de enlaces del documento, actuando como el elemento raíz del árbol de sincronización.

El *Módulo de Sincronización* describe la presentación de elementos SMIL a lo largo del tiempo. Sus elementos principales son tres. El elemento `seq` permite reproducir varios elementos en secuencia. El elemento `excl` permite reproducir varios elementos en serie pero sin imponer ningún orden. Por último, el elemento `par` permite reproducir varios elementos como un grupo (en paralelo). Nótese que estos elementos lo único que hacen es capturar la semántica impuesta por los nodos que forman los componentes compuestos del modelo Amsterdam (Figura 2.5 del Capítulo 2). De esta forma si queremos indicar que tenemos una presentación formada por un vídeo y un clip de audio que van a ser mostrados durante 30 sg. no tenemos más que considerar el siguiente documento SMIL,

```
<smil xmlns="xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
    <par>
      <video src="intro.mpg" dur="30s"/>
      <audio src="intro.au" dur="30s"/>
    </par>
  </body>
</smil>
```

Los *Módulos de diseño* permiten especificar la distribución espacial de los elementos SMIL en la presentación. Con este fin se utiliza el elemento `layout` para posicionar en una superficie visual de presentación (visual o acústica) los objetos declarados en el elemento `body` del documento SMIL. De no especificarse este elemento, el posicionamiento de los objetos declarados dependerían de la implementación. El elemento `region` se utiliza para definir las regiones concretas por las que va a estar formada una presentación. De esta forma si deseamos una presentación formada por una foto y un texto en una misma ventana, en la posición especificada por los atributos `top` (offset respecto del borde superior) y `left` (offset respecto del borde izquierdo), durante treinta segundos, no tendríamos más que especificar (el ejemplo anterior era dependiente de la implementación, mientras que este no lo es),

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
    <head>
      <layout>
        <region id="foto"  top=”0” left=”0” width="100" height="100"/>
```

```
        <region id="texto" top=”0” left=”150” width="100" height="100"/>
      </layout>
    </head>
    <body>
      <par>
        <img region=”foto” src="foto.mpg" dur="30s"/>
        <text region=”texto” src="texto.txt" dur="30s"/>
      </par>
    </body>
  </smil>
```

SMIL además permite tener varias ventanas activas en la misma presentación. Para ello utiliza el elemento `topLayout` (dentro del elemento `layout`) por cada ventana activa que se desee tener en cada presentación. A su vez el elemento `topLayout` puede estar formado por elementos `region`.

Los *Módulos de enlaces* permiten establecer enlaces entre los componentes de un documento SMIL. Al igual que con el módulo de sincronización, su finalidad es capturar los enlaces descritos por el modelo Amsterdam (ver Figura 2.7 del Capítulo 2). SMIL 2.0 permite definir solamente enlaces unidireccionales *in-line* (es decir, aquellos cuyas anclas están definidas en los propios componentes que constituyen el enlace, como en el caso del elemento `a` de HTML). Está formado por tres módulos. El *Módulo de Atributos de Enlace* incluye un conjunto de atributos utilizados para proporcionar la semántica de enlace a los elementos enlazados. El *Módulo de Enlace Básico* incluye los elementos de enlace SMIL. El *Módulo de Enlazado de Objetos* incluye características adicionales de enlace que ciertos perfiles del lenguaje podrían no incluir. Los elementos principales para definir enlaces entre objetos SMIL son `a` y `area`. El elemento `a` permite definir objeto como el ancla origen de un enlace. El elemento `area` permite definir una porción de un objeto como el ancla origen de un enlace. De esta forma si queremos indicar que al seleccionar una imagen aparezca otra presentación SMIL no tenemos más que escribir,

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
   <a href="otraPresentación.smil">
     <img src="foto.jpg" dur="indefinite"/>
   </a>
 </body>
</smil>
```

Si lo que deseamos es indicar que solamente una región de esa foto es el ancla origen del enlace, no tenemos más que escribir,

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <body>
    <img src="foto.jpg" dur=”indefinite”>
     <area shape="rect" coords="0%,0%,50%,50%"
                        href="otraPresentación.smil"/>
    </img>
  </body
</smil>
```

Si estos son los mecanismos básicos de enlace ¿cómo podemos expresar en SMIL la existencia de enlaces entre textos? No olvidemos que, en principio, multimedia no prohibe la aparición de texto. Pues bien hemos encontrado el talón de Aquiles de SMIL: los enlaces con origen en un texto. Supongamos que tenemos un texto en el cual aparecen enlaces con varios textos. Si optamos por utilizar el elemento `text` de SMIL, los enlaces con origen en dicho elemento deberían especificarse a través del atributo `area`, es decir, como si de una imagen se tratase. Si optamos por sustituir el elemento `text` por una hoja HTML con los enlaces HTML incorporados tenemos dos opciones: suponer que los enlaces son a nivel HTML o a nivel SMIL. En el primer caso la activación de enlaces solo puede afectar al documento HTML sin afectar a la presentación SMIL (por ejemplo, no podemos simular los marcos HTML con `regions` SMIL y enlaces HTML). Si optamos por considerar los enlaces HTML como enlaces SMIL debemos “elevar” los enlaces de los contenidos HTML a enlaces SMIL. La forma de conseguir esto es mediante el atributo `fragment` del elemento `area`, perteneciendo este atributo al módulo de enlazado de objetos, opcional en el lenguaje. Supongamos por tanto que queremos una presentación SMIL similar al ejemplo de los marcos vistos en el apartado de HTML (4.3.2). En este caso el archivo con el índice sería:

```
<html>
  <head>
```

```
    <title>Índice</title>
  </head>
  <body>
    <ul>
     <li><a name="enlace1" href="cont1.htm">Contenido 1</a></li>
     <li><a name="enlace2" href="cont2.htm">Contenido 2</a></li>
    </ul>
   </body>
</html>
```

y el documento SMIL sería:

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
    <head>
      <layout>
        <region id="indice"     width="100" height="100"/>
        <region id="contenidos" width="100" top   ="100"/>
      </layout>
    </head>
    <body>
      <par>
        <text region="indice" src="indice.html" dur="indefinite">
          <area fragment="enlace1" href="#EnlaceUno"/>
          <area fragment="enlace2" href="#EnlaceDos"/>
        </text>
        <excl dur="indefinite" >
          <text id="EnlaceUno"
                region="contenidos"
                src="cont1.html" dur="indefinite"/>
          <text id="EnlaceDos"
                region="contenidos"
                src="otroContenido2.html" dur="indefinite"/>
        </excl>
      </par>
    </body>
  </smil>
```

Ahora los enlaces HTML son considerados como enlaces SMIL. Nótese que en el caso de segundo enlace del índice hemos cambiado el destino inicial de `cont2.html` a `otroContenido2.html`. Esta "elevación" de enlaces del nivel de contenidos (HTML) al nivel de presentación (SMIL) será un componente estándar del modelo hipermedia presentado en esta tesis: la *función de canalización de enlaces* (apartado 5.2.3 del Capítulo 5), aunque en dicho modelo las anclas del nivel de contenidos son direccionables directamente desde el nivel de presentación. Nótese que en SMIL, a diferencia de HTML, los enlaces si que son independientes del esquema navegacional que se desee imponer sobre ellos, pero a costa de redefinir todas las anclas de contenidos a nivel navegacional. Al igual que en HTML, y debido a la imposibilidad de indicar la carga dinámica de contenidos en una presentación SMIL, no es posible variar dinámicamente el contenido por defecto asociado a una región de una presentación SMIL cuando se carga por primera vez dicha presentación.

Nos falta por último indicar que sucede cuando un enlace SMIL tiene como destino parte de un elemento SMIL que está siendo reproducido. Dicho de otra forma, veamos como SMIL determina sin ningún género de duda el resultado de enlaces intercontextuales que vimos en el modelo Amsterdam (sección 2.3.1.1). Esta semántica viene especificada en el módulo temporal y de sincronización. Básicamente viene a decir que: (i) si el elemento destino está activo se vuelve al momento más cercano especificado por el ancla; (ii) si se puede determinar la temporización asociada al elemento, se reproduce a partir del punto más cercano especificado por el ancla; (iii) si no se puede determinar la temporización asociada al elemento, se intenta determinar para sus ancestros, tomando como tiempo de comparación con el valor especificado por el ancla el calculado como resultado de este proceso de cálculo, o el del padre, en caso de que el proceso de cálculo no tenga éxito.

4.4.3 El Lenguaje Extensible de Interfaz de Usuario (XUL)

Introducción

El lenguaje XUL (*XML-based User-interface Language*, pronunciado *zuul* y que rima con *cool* [Deakin 01]) fue creado para la aplicación Mozilla (un navegador Web creado en el seno de Netscape) con el fin de describir su interfaz de usuario. Por definición tanto el navegador, como el propio lenguaje están en continua evolución [Deakin 01]). La versión aquí presentada se corresponde con el tutorial del 24 de septiembre del 2001 [Deakin 01]. Una muestra de este continuo devenir es que el último navegador comercializado por Netscape (la versión 6.0) no proporciona soporte al lenguaje XUL.

La finalidad de XUL es la descripción de las interfaces de usuario clásicas de las aplicaciones que se ejecutan en un navegador utilizando un lenguaje de marcas. Es decir, en vez de construir la interfaz de una aplicación programando directamente el API (por ejemplo Swing en Java) se procede a su descripción utilizando un documento XML. Esta aproximación varía del uso de una herramienta de descripción visual de interfaces (por ejemplo, Forte for Java de Sun Microsystems) en que la herramienta visual produce código fuente que al ser ejecutado proporciona la interfaz de usuario, y el documento XML debe ser interpretado por el navegador para poder obtener la interfaz. Si intentamos asimilar esta aproximación al marco *Modelo-Vista-Controlador* (es decir, lógica de la aplicación, componente presentacional de la interfaz de usuario que se impone sobre la lógica, y respuesta a los eventos (en su acepción habitual en programación) que se producen en dicha interfaz actuando sobre la lógica de la aplicación), XUL (como lenguaje de marcas) solo cubre la parte *Vista* del marco anterior, quedando encomendada la parte *Modelo-Controlador* a código JavaScript.

Como podemos observar la aproximación XUL representa un cambio de filosofía con respecto a los lenguajes HTML y SMIL. Estos se centraban en un dominio de aplicación de la informática (hipermedia y multimedia, respectivamente) y proporcionaban un mecanismo de descripción de aplicaciones de esa área. Por el contrario, XUL trasciende la aplicación a un dominio concreto, centrándose en la descripción de una componente computacional (la interfaz de usuario) aplicable a distintos dominios de la informática. La razón de incluir XUL en el estado del arte de esta tesis es doble. Por un lado comprobar como los lenguajes de marcas pueden utilizarse para describir interfaces de usuario (cosa que por otro lado ya hacían HTML y SMIL, pero en dominios más concretos), y por otro estudiar su aplicabilidad a la construcción de aplicaciones hipermedia.

La mayor ventaja derivada de la aproximación XUL, y que fue la razón de su aplicación en el seno del proyecto Mozilla, es la facilidad para ser utilizado en plataformas heterogéneas. En efecto, con independencia del sistema sobre el que XUL vaya a ejecutarse (como siempre en estos casos, el dependiente será el intérprete, máquina virtual, navegador o como desee denominarse a este tipo de ingenios), el lenguaje permite definir la mayoría de los elementos encontrados en las interfaces gráficas de usuario. Entre estos cabe citar: controladores de entrada (como cajas de texto), barras de herramientas con botones, menús, barras de menús, menús desplegables, etc. Además, los contenidos mostrados dentro de estos elementos pueden provenir de archivos XUL o de archivos en formato RDF [RDF].

Descripción técnica

El elemento raíz de un documento XUL es el elemento `window` (en vez del elemento XUL, tal y como estamos acostumbrados tradicionalmente en el ámbito de los lenguajes de marcado).
Consecuencia directa de este hecho es que cada documento XUL solo puede contener la descripción de una ventana. Así, mediante el documento:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window id="ventana" title="Ventana de prueba"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
</window>
```

describimos la interfaz una aplicación formada por una única ventana y que no es capaz de responder a ningún evento. En el ejemplo anterior, la instrucción de procesamiento `xml-stylesheet` informa al navegador sobre la apariencia concreta de los elementos que componen la interfaz de usuario. Si por ejemplo deseamos añadir dos botones a la ventana anterior no tenemos más que incluir dos apariciones del elemento `button`. En efecto con el siguiente documentos:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window id="ventana" title="Ventana de prueba"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <button id="boton1" label="Aceptar" default="true"/>
```

```
  <button id="botón2" label="Cancelar"/>
</window>
```

conseguimos la inclusión de dichos botones en la pantalla (la cual sigue siendo incapaz de responder a ningún evento).

En cuanto a la inclusión de paneles de contenido dentro de una ventana el lenguaje proporciona el elemento `iframe`, que permite especificar los marcos en los que se divide una ventana. Por ejemplo, si deseamos dividir una ventana en tres marcos no tenemos más que escribir:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window id="ventana" title="Ventana de prueba"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <box>
    <iframe id="marco1" width="60" height="20" src="cont1.html"/>
    <iframe id="marco2" width="60" height="20" src="cont2.html"/>
    <iframe id="marco3" width="60" height="20" src="cont3.html"/>
  </box>
</window>
```

donde el elemento `box` permite definir el *layout* de los marcos dentro de una ventana. Por cierto, el ejemplo anterior no funciona correctamente en la versión 0.9.4 del navegador Mozilla.

A diferencia de HTML y SMILE lenguajes definidos para un dominio concreto, XUL es genérico y necesita por tanto la capacidad de responder a eventos de carácter general. El mecanismo para tratar eventos genéricos en XUL va a ser mediante JavaScript, introducido en el documento (bien por inclusión o por referencia) a través del elemento `script`. De esta forma si queremos asociar código JavaScript a una ventana XUL para que proporcione una respuesta adecuada a los eventos allí generados no tenemos más que añadir la siguiente línea justo debajo de la etiqueta de apertura de `window`, antes de cualquier otro elemento.

```
<script src="controlEventos.js"/>
```

Nótese que si deseamos utilizar XUL en el contexto de creación de aplicaciones hipermedia puede hacerse en combinación con los lenguajes HTML y JavaScript. Consideremos por ejemplo el caso que no podían resolver ni HTML ni SMIL. Tenemos un texto que habla sobre el Mundo, enlazando con otros textos sobre países. A su vez estos textos sobre países enlazan con textos sobre ciudades. Si deseamos presentar el texto sobre el mundo en una ventana, y el texto sobre los países y ciudades en otra ventana, vimos que en HTML y SMIL debido a la imposibilidad de cargar dinámicamente contenidos en marcos, cuando estos son invocados por primera vez, necesitaríamos tantos documentos como países tuviésemos. Pues bien, ahora en XUL no hace falta más que considerar dos ventanas, la segunda de ellas con dos marcos para especificar el esquema navegacional. Por otro lado, para los contenidos podemos considerar páginas HTML convenientemente enlazadas. Asignamos a la primera ventana la página HTML con el texto sobre el mundo, junto a código JavaScript. Cada vez que se seleccione un enlace con origen en el texto del mundo, el código JavaScript lo detecta y carga la ventana XUL el país correspondiente (disponemos de la función JavaScript `loadPage()` [Mozilla XUL], capaz de hacer esto). La selección de enlaces entre países y ciudades que se presentan en la segunda ventana requeriría un tratamiento similar. La cuestión en esta aproximación es: ¿cómo sabe el código JavaScript en que marco concreto debe mostrar los contenidos seleccionados por un ancla origen como destino de un hiperenlace? Básicamente hay dos formas: (i) las anclas HTML incluyen información sobre el marco a través del atributo `target`, sirviendo para distinguir marcos de distintas ventanas (en contraposición a la aproximación clásica HTML); (ii) el código JavaScript tiene una lista que le indica donde mostrar cada contenido en respuesta a la selección de un enlace. La primera solución invalida la separación de contenidos y esquema navegacional. La segunda requiere un tercer documento que liste para cada ancla su enlace destino, con los problemas de consistencia de la información, y de trabajo añadido que representa. Nótese que esta aproximación también puede aplicarse a marcos HTML sin necesidad de utilizar XUL.

El problema de indicar en que marco concreto debe mostrarse los contenidos como respuesta de un hiperenlace aparece resuelto en el modelo hipermedia propuesto en el Capítulo 5 de esta tesis mediante la *función de canalización de enlaces* (sección 5.2.3) y la *semántica de presentación* (sección 5.2.4). Además en lenguaje propuesto para la construcción de prototipos automáticamente presentado en el Capítulo 6, la especificación de esta canalización se hace al mismo tiempo que se asignan contenidos a los marcos, facilitando así la construcción de la aplicación.

4.4.4 El lenguaje de marcado de interfaz de usuario (UIML)

Introducción

El lenguaje UIML (*User Interface Markup Language*) [Abrams 00], [Harmonia UIML] es un lenguaje para la descripción de interfaces gráficas de usuario. A diferencia de XUL, estas interfaces no van a ser de aplicaciones interpretadas por un navegador, sino aplicaciones computacionales de propósito general. Además, ahora, se consideran interfaces de usuario para distintos lenguajes (Java, HTML, WML [WML], etc.). Aunque al igual que XUL el propósito de UIML es la independencia de la plataforma, en la práctica los documentos UIML son totalmente dependientes de la plataforma. En efecto, a diferencia de la aproximación XUL donde los documentos están formados por descripciones de la interfaz de usuario en términos abstractos, independientes de cualquier plataforma concreta (ventana, marco, botón, etc.), los documentos UIML están formados por una mera enumeración de "estructuras" de la interfaz de usuario, ligando esas estructuras a entes concretos pertenecientes a las interfaces de usuario del lenguaje destino. La ventaja de esta aproximación, es que un elemento ventana podría no ser útil en todos las plataformas, como en los PDAs. Eso si, a diferencia de XUL, las descripciones UIML no son interpretadas por un navegador, sino traducidas a código fuente para ser compilado (salvo en el caso de HTML, en cuyo caso es interpretado).

En nuestro caso, como las aplicaciones hipermedia por definición suponen una serie de recursos computacionales mínimos para ejecutarse, utilizaremos una descripción del esquema navegacional (o interfaz de usuario) impuesto sobre los contenidos en términos similares a la descripción XUL. La *DTD de la aplicación*, descrita en la sección 6.3.2 del Capítulo 6 define el vocabulario que utilizaremos.

Descripción técnica

En este caso concreto la descripción técnica tiene cierta complejidad, ya que el lenguaje no se encuentra en un estado estable, habiendo información contradictoria en función de la fuente consultada. De todas formas intentaremos hacer una descripción lo más correcta posible en función a la última evolución del lenguaje presentada.

Un documento uiml (`uiml`) está formado básicamente por metainformación (`head`), la descripción de la interfaz (`interface`), y el vocabulario permitido dentro de esa descripción de la interfaz (`peers`). Como ya hemos comentado en UIML la interfaz de usuario está formada por la estructura de la misma (`structure`), dividida a su vez en distintos componentes o partes (`part`). Así si queremos describir una interfaz Java formada por un `JFrame` que contiene un `JButton`, no tenemos más que indicar:

```
<uiml>
  <interface>
    <structure>
      <part name="JF" class="JFrame">
        <part name="JB" class="JButton"/>
      </part>
    </structure>
  </interface>
</uiml>
```

Parece razonable suponer que no solo es necesario definir que partes componen una interfaz de usuario, sino las propiedades que van a caracterizar a dichas partes. La forma de conseguir esto es mediante el elemento `style`, que indica las propiedades (`property`) de cada parte de la interfaz de usuario. Así si queremos indicar el título de la ventana anterior y el texto del botón, no tenemos más que considerar el siguiente documento:

```
<uiml>
  <interface>
    <structure>
      <part name="JF" class="JFrame">
        <part name="JB" class="JButton"/>
      </part>
    </structure>
    <style>
      <property
```

```
  </interface>
  <style>
    <property part-name="JF" name="title">Esto es un JFrame</property>
    <property part-name="JB" name="text">Aceptar</property>
  </style>
</uiml>
```

Hemos dicho que estamos generando la interfaz de usuario para un programa Java. En algún lugar será necesario indicar con que clases concretas de Java (del API Swing, por ejemplo) se corresponden las partes de la interfaz. Para esto tenemos el elemento `peers`, el cual formado por presentaciones (`presentation`) que ligan los componentes de la interfaz gráfica descrita en Swing con entes concretos del lenguaje destino (clases Swing en este caso) a través del elemento `d-class`. También es posible considerar sin contenido al elemento `peers`, y referenciarlo a través de su atributo `source`. A su vez este elemento se encarga de ligar las propiedades descritas en el documento con las propiedades de los entes reales. Así para indicar que la parte `JFrame` se corresponde con un `JFrame` Swing y que la parte `JButton` se corresponde con un `JButton` Swing no tenemos más que considerar el siguiente fragmento:

```
<uiml>
  <interface>
    <structure>
      <part name="JF" class="JFrame">
        <part name="JB" class="JButton"/>
      </part>
    </structure>
    <style>
      <property
  </interface>
  <style>
    <property part-name="JF" name="title">Esto es un JFrame</property>
    <property part-name="JB" name="text">Aceptar</property>
  </style>
  <peers>
    <presentation name="java-jfc"
                  source="http://uiml.org/toolkits/JavaJFC2.uiml">
      <d-class name="JFrame"
               maps-type="method" maps-to"javax.swing.JFrame">
        <d-property name="title" maps-type="setMethod" maps-to"setText">
           <d-param type="String"/>
        </dproperty>
        .....
      </d-class>
      <d-class name="Button"
               maps-type="method" maps-to"javax.swing.JButton">
        <d-property name="text" maps-type="setMethod" maps-to"setTitle">
           <d-param type="String"/>
        </dproperty>
        .....
      </d-class>
    </presentation>
</uiml>
```

Como UIML es un lenguaje para describir interfaces genéricas es necesario indicar cual es la respuesta a eventos genéricos. Con este fin el elemento `interface` contiene un elemento `behaviour` formado a su vez por un conjunto de reglas (`rule`). Cada regla contiene una condición (`condition`) y una acción (`action`), de tal forma que cuando la condición se cumple se lleva a cabo la acción. A su vez la condición puede hacerse cierta cuando sucede un evento (`event`) o cuando sucede un evento y el valor de algún dato asociado con el evento es igual a un cierto valor (`equal`). Las acciones pueden ser variaciones en las propiedades de algún elemento (`property`), la invocación de un método o función (`call`) o el disparo de otro evento (`event`). Para estos dos últimos casos UIML no especifica como se lleva a cabo la llamada (por ejemplo, RPC, RMI, CORBA). Al igual que las partes de la interfaz, los eventos también se tratan como nombres de clases. El elemento `d-class` especifica la correspondencia entre clases de eventos a clases en el lenguaje destino.

El intentar describir una aplicación hipermedia utilizando UIML tendría una dificultad similar a intentar describirla utilizando un lenguaje de computación de propósito general (por ejemplo, Java), ya que lo único que describiría es la interfaz de la aplicación, y parte del comportamiento de la misma. Evidentemente si volvemos al marco Modelo-Vista-Controlador, nos falta cuanto menos toda la parte Modelo, es decir, la

lógica de la aplicación. Por tanto no nos parece adecuado el uso de UIML como mecanismo de descripción de esquemas navegacionales de aplicaciones hipermedia. Nótese que esto no es una crítica al lenguaje, ya que el uso del mismo no tiene por interés el restringirse a esta área de la informática.

4.5 Estándares relacionados con XML

Estudiaremos en este apartado una serie de estándares relacionados con XML cuya finalidad última es el tratamiento de documentos marcados. Esta es una de las mayores ventajas de XML frente a SGML: disponer de un conjunto de herramientas estándar que faciliten el tratamiento de los documentos marcados. Hemos decidido incluirlos debido al uso que se va a hacer de ellos en esta tesis (salvo SAX, que se incluyó por razones de completitud). Debido a su menor interés hipermedia, a excepción de XLink, proporcionaremos una descripción muy breve de cada uno.

4.5.1 El lenguaje de localización XML (XPath) y el lenguaje señalizador XML (XPointer)

XPath

XPath [W3C XPath] es un estándar W3C para direccionar partes de un documento XML. Se diseñó para que formase parte de XPointer (el poder expresivo de ambos es similar al proporcionado por el módulo de direcciones de localización HyTime aunque sin utilizar arquitecturas) y de XSLT. El uso de este estándar (o recomendación) que hace la aproximación presentada en el Capítulo 6 de esta tesis, será el de relacionar el esquema navegacional de una aplicación hipermedia con sus contenidos (es decir, para implementar la función de asignación de contenidos **d** de Pipe).

XPath modela un documento XML como un árbol de nodos. Básicamente se corresponde con el árbol de análisis clásico del documento XML donde los atributos también aparecen como hijos de los elementos, aunque a través de una relación distinta de la establecida entre elementos. Podríamos decir que en:

```
<padre atributo="hola"><hijo>Mundo</hijo></padre>
```

tenemos `hijoEstructural(padre, hijo)`, `hijoAtributo(padre, atributo)`. En vez de esta sencilla distinción el estándar recurre a una solución más rocambolesca (apartado 5.3 del estándar): "*Each element node has an associated set of attribute nodes; the* ***element is the parent of each of these attribute nodes****; however, an* ***attribute node is not a child of its parent element***".

Las expresiones XPath determinan *caminos de localización* sobre este árbol, estando formados estos caminos por los *pasos de localización*. Un paso de localización tiene tres partes:

- Un *eje*, que especifica la relación entre los nodos seleccionados por el paso de localización y el *nodo contexto* (básicamente desde donde se inicia la búsqueda). En la práctica suele obviarse, ya que el eje por defecto que se supone es `child`, es decir, los hijos del nodo contexto.
- Un *nodo prueba*, que especifica el tipo de nodo y el nombre de los nodos seleccionados por el paso de localización. Es decir, el nombre del componente del árbol que queremos localizar.
- Cero o más *predicados*, que utilizan expresiones para refinar el conjunto de nodos seleccionados por el paso de localización.

Así si queremos seleccionar todos los `párrafo`s escritos por `antonio` (descrito a través del atributo `autor` de `párrafo`), que se encuentran el `capítulo` tercero (en cuanto a relación con su padre, si el orden de los capítulos no coincide con el orden jerárquico en el árbol XML, la pregunta siguiente no devolvería el tercer capítulo "real") de un `libro`, no tenemos más que escribir la expresión XPath (camino de localización):

```
/libro/capítulo[3]/parrafo[@autor="antonio"]
```

Donde hemos omitido la información sobre ejes, tenemos tres pasos de localización, el primero formado por un nodo de prueba (`libro`), el segundo por un nodo de prueba y un predicado (`capítulo[3]`) y el tercero también formado por un nodo de prueba y un predicado (`parrafo[@autor="antonio"]`).

Como podemos comprobar, en la práctica, las expresiones XPath se asemejan bastante a las rutas (o *paths*) de los sistemas operativos. La razón es evidente: ambas expresiones sirven para localizar objetos en una estructura arbórea.

XPointer

XPointer [W3C XPointer] es un desarrollo W3C utilizado para localizar partes de documentos XML que, en contraposición a XPath, no tienen porque corresponderse con nodos "enteros". La necesidad de este estándar es clara. Si queremos utilizarlo junto con XLink para localizar anclas de enlaces en documentos, necesitamos mayor granularidad para seleccionar partes de documentos (de forma similar a lo que permitía el módulo de direcciones de localización HyTime, aunque sin utilizar arquitecturas). En el momento de escribir esta tesis XPointer no tiene la calificación de *W3C Recommendation*, estando en estos momentos en *W3C Candidate Recommendation 11 September 2001*. Este hecho no se debe a falta de madurez del estándar, sino a ciertos problemas con la patente *Method and system for implementing hypertext scroll attributes* que posee Sun Microsystems [Dodds 01b].

XPointer es una extensión de XPath que permite:
- Direccionar puntos y rangos, además de nodos enteros.
- Localizar información por ajuste de cadenas.
- Utilizar expresiones de direccionamiento en referencias URI (*Uniform Resource Identifier*) como identificadores de fragmentos.

Así, si queremos localizar la primera ocurrencia de la palabra `externocleidomastoideo` en el segundo `párrafo` escrito por `antonio` (descrito a través del atributo `autor` de `párrafo`), que se encuentran el `capítulo` tercero (en cuanto a relación con su padre, si el orden de los capítulos no coincide con el orden jerárquico en el árbol XML, la pregunta siguiente no devolvería el tercer capítulo "real") de un `libro`, no tenemos más que escribir la expresión XPointer:

```
xpointer(string-range(/libro/capítulo[3]/parrafo[@autor="antonio"][2],
                      "externocleidomastoideo")[1])
```

4.5.2 El lenguaje de enlace XML (XLink)

XLink [W3C XLink] permite especificar enlaces entre documentos XML de manera similar a como lo hacía el módulo de hiperenlaces HyTime, pero sin utilizar arquitecturas. La manera de indicar que un elemento genérico tiene un enlace XLink es incluyendo el atributo `xlink:type` en la declaración de dicho elemento. Cualquier otra información que necesite incluirse se indica mediante un atributo precedido por el espacio de nombre `xlink` [W3C namespaces]. En contraposición a HyTime no se espera un tratamiento para obtener el grove arquitectónico. Simplemente una aplicación capaz de procesar enlaces XLink, responderá de manera adecuada al ver estos atributos en los elementos (tal y como funciona el generador automático de aplicaciones propuesto en el Capítulo 6 de esta tesis). Este tratamiento más sencillo presenta la desventaja de una presentación menos formal del estándar.

XLink tiene dos tipos de enlaces: *extendidos* y *simples*. Los primeros permiten definir enlaces n-arios independientes de las anclas (como los enlaces `hylink` de HyTime, y en particular los de tipo `varlink`), mientras que los segundos permiten definir enlaces unarios dependientes de las anclas (como los enlaces `clink` de HyTime, o los basados en `a` HTML). En ambos casos las localizaciones direccionadas pueden especificarse utilizando expresiones XPointer.

Enlaces extendidos

Como ya hemos comentado permiten especificar enlaces n-arios independientes de las anclas. Estos enlaces pueden contener una mezcla de los siguientes elementos, en cualquier orden, y posiblemente mezclados con otro contenido y marcado.
- Elementos `locator` que direccionan los recursos remotos que participan en el enlace. Es decir, permite enlaces con anclas localizadas en documentos externos.

- Elementos `resource` que identifican los recursos locales que participan en el enlace. Es decir, permite enlaces con anclas localizadas en el propio documento.
- Elementos `arc` que proporcionan las reglas de navegación entre los recursos enlazados.
- Elementos `title` que proporcionan información sobre los enlaces.

Así si por ejemplo en la siguiente información:

```
... escritas por Navarro en su ...
```

deseamos indicar que `Navarro` es el ancla origen de un enlace n-ario a varias biografías suyas, debemos incluir la siguiente información en la DTD:

```
<!ELEMENT autor (info|origen|destino|dirección)*>
<!ATTLIST autor
        xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
        xlink:type (extended) #FIXED "extended"
        xlink:role CDATA #IMPLIED
        xlink:title    CDATA #IMPLIED>

<!ELEMENT info (#PCDATA)>
<!ATTLIST info
        xlink:type (title) #FIXED "title"
        xml:lang CDATA #IMPLIED>

<!ELEMENT origen (#PCDATA)>
<!ATTLIST origen
        xlink:type (resource) #FIXED "resource"
        xlink:role CDATA #FIXED "persona"
        xlink:title CDATA #IMPLIED
        xlink:label NMTOKEN #IMPLIED>

<!ELEMENT destino EMPTY>
<!ATTLIST destino
        xlink:type (locator) #FIXED "locator"
        xlink:href CDATA #REQUIRED
        xlink:role CDATA #FIXED "biografía"
        xlink:title CDATA #IMPLIED
        xlink:label NMTOKEN #IMPLIED>

<!ELEMENT dirección EMPTY>
<!ATTLIST dirección
        xlink:type (arc) #FIXED "arc"
        xlink:arcrole CDATA #IMPLIED
        xlink:title CDATA #IMPLIED
        xlink:show (new|replace|embed|other|none) #IMPLIED
        xlink:actuate (onLoad|onRequest|other|none) #IMPLIED
        xlink:from NMTOKEN #IMPLIED
        xlink:to NMTOKEN #IMPLIED>
```

Ahora la información se representa de la siguiente manera:

```
... escrito por
    <autor>
      <info>enlace n-ario de un autor con sus biografías</info>
        <origen xlink:label="navarro">Navarro</origen>
        <destino
               xlink:href="biografias/navarro1.xml"
               xlink:label="bioNav1"/>
        <destino
               xlink:href="biografias/navarro2.xml"
               xlink:label="bioNav2"/>
        <dirección
               xlink:from="navarro"
               xlink:to="bioNav1"/>
        <dirección
               xlink:from="navarro"
               xlink:to="bioNav2"/>
    <autor>
    en su ...
```

Lo normal en estos enlaces es que todas las anclas sean externas. Para conseguir esto basta con cambiar `origen` de `resource` a `locator`, pudiendo colocar el enlace en un documento externo (al estilo HyTime).

Enlaces simples

Este tipo de enlaces (unidireccionales, unarios, y con el ancla origen interna) pueden ser representados mediante enlaces extendidos, utilizándose solamente para facilitar su representación. Así si en el ejemplo anterior decidimos que cada autor solo enlace con una biografía, debemos incluir en la DTD:

```
<!ELEMENT autor (#PCDATA)>
<!ATTLIST autor
        xlink:type (simple) #FIXED "simple"
        xlink:href CDATA #IMPLIED
        xlink:role NMTOKEN #FIXED "persona"
        xlink:arcrole CDATA #IMPLIED
        xlink:title CDATA #IMPLIED
        xlink:show (new|replace|embed|other|none) #IMPLIED
        xlink:actuate (onLoad|onRequest|other|none) #IMPLIED>
```

Para incluir el enlace en el documento no tenemos más que:

```
... escrito por <autor xmlink:href="biografia1.xml">Navarro</autor> en su ...
```

4.5.3 Transformaciones XSL (XSLT)

Según la recomendación W3C, XSLT [W3C XSLT] es un mecanismo para transformar documentos XML en documentos XML (realmente hay una excepción: la transformación vacía devuelve un texto sin etiquetas, que no es un documento XML). XSLT es parte del Lenguaje Extensible de Hojas de estilo, XSL [W3C XSL]. Las transformaciones XSLT serán utilizadas por el generador automático de prototipos descrito en el Capítulo 6 para pasar de contenidos XML a páginas HTML que serán utilizadas por `JEditorPane`s Swing con el fin de proporcionar el esquema navegacional a los contenidos.

Una cuestión interesante de la sintaxis XSLT es que se expresa en sintaxis XML, lo cual obliga a utilizar constantemente en espacio de nombres `xsl` para distinguir las etiquetas XSLT de las etiquetas del documento destino. Nótese que esta aproximación no encaja con la visión proporcionada en el apartado 4.2 de este mismo capítulo (proporcionar lenguajes de marcas cuando a partir de los terminales del lenguaje objeto no pueda identificarse una sintaxis sobre la cual especificar un tratamiento), y es cuanto menos discutible.

XSLT opera sobre la estructura arbórea definida por XPath. La forma intuitiva de funcionar de XSLT es la siguiente: se inicia un recorrido en preorden por el árbol XPath del documento origen, buscándose una plantilla XSLT (`xsl:template`) que se ajuste a cada elemento del árbol. Dichas plantillas se ajustan a los elementos del árbol origen a través de una expresión XPath. Cada plantilla proporciona la estructura de salida del procesamiento indicando que subpartes de los nodos localizados deben ser procesadas y cómo, pudiendo variar el recorrido en preorden del árbol origen. Cada vez que se procesan las subpartes de estos nodos, se sigue buscando la existencia de plantillas XSLT que indiquen su procesamiento. Si las hay se aplican. Si no, se aplica el procesamiento por defecto, que consiste en recorrer todos los subelementos de un nodo, extrayendo su contenido `#PCDATA`.

De esta forma si tenemos el documento XML:

```
<a>
  <b>Hola</b>
  <c>Mundo</c>
  <d>Mundial</d>
</a>
```

y queremos obtener el documento:

```
(a)
```

```
 (c)Mundo(c/)
 (b)Hola(/b)
(/a)
```

No tenemos más que aplicar la transformación XSLT:

```
<xsl:template match="a">
  (a)
    <xsl:apply-templates select="b"/>
    <xsl:apply-templates select="c"/>
  (/a)
</xsl:template>

<xsl:template match="b">
  (b)
    <xsl:apply-templates/>
  (/b)
</xsl:template>

<xsl:template match="c">
  (c)
    <xsl:apply-templates/>
  (/c)
</xsl:template>
```

XSLT es un lenguaje potente y el ejemplo proporcionado es una pequeña muestra de sus posibilidades expresivas. Para una descripción completa del lenguaje, la mejor opción, según nuestro entender, es el propio estándar [W3C XSLT]. Además en algunos entornos [Dodds 01a], XSLT, (o incluso el propio XPath), puede considerarse como un lenguaje de *query*, en sustitución de estándares más específicos como XML Query [W3C Query].

4.5.4 El modelo de objetos de documentos (DOM) y la interfaz de programación de aplicaciones simple para XML (SAX)

DOM

El modelo de objetos de documento [W3C DOM] (nos restringimos al nivel 1 porque es el que utilizaremos en la tesis), es un estándar que define un conjunto de interfaces para crear, acceder y manipular estructuras internas de documentos XML. Aunque parezca chocante el modelo DOM XML no coincide con el modelo DOM HTML. El interfaz DOM será utilizado por el generador automático de prototipos descrito en el Capítulo 6 de esta tesis para generar prototipos Java de aplicaciones hipermedia a partir de descripciones XML. Hemos preferido DOM en vez de SAX debido a que es estándar W3C, y desde nuestro punto de vista, proporciona una programación mucho más sencilla e intuitiva. Entre los interfaces descritos por DOM distinguimos:

Nombre de la clase de interfaz	Descripción
`Document`	Documento XML completo
`Node`	Nodo XML (elementos, atributos, texto...)
`NodeList`	Lista de nodos
`NamedNodeMap`	Lista de nodos accesibles por nombre
`Element`	Nodo elemento
`Attr`	Nodo atributo
`Text`	Contenido de un nodo elemento o atributo

Así si queremos un programa que tome como entrada el documento

```
<?xml version="1.0" ?>
  <a at1="a1" at2="a2">
  <b>hola</b>
  mundo
  <c at1="a1">
    mundial
   <d>mundoso</d>
  </c>
```

```
</a>
```

y produzca:

```
(documento)

(a at1= a1 at2= a2)
  (b)hola(/b)
  mundo
  (c at1= a1)
    mundial
    (d)mundoso(/d)
  (/c)
(/a)
```

solo tenemos que considerar la clase `EjemploDOM` siguiente:

```
public class EjemploDOM {

public static void procesar (Node nodo)
    {   String nomEl=null;
        Node hijo;

        switch (nodo.getNodeType())
        {
          case Node.DOCUMENT_NODE: { System.out.print("(documento)");
                                     break; }
          case Node.ELEMENT_NODE: { nomEl= procesarElemento(nodo);
                                    break; }
          case Node.TEXT_NODE: { System.out.print(nodo.getNodeValue());
                                 break; }
            case Node.DOCUMENT_TYPE_NODE:
                       { System.out.print("DOCTYPE "+nodo.getNodeName());
                         break; }
          default:
            System.out.print("? "+nodo.getNodeName());
        }
        for (hijo = nodo.getFirstChild();
                hijo != null; hijo = hijo.getNextSibling())
              procesar(hijo);
        if (nomEl != null) System.out.print("(/"+nomEl+")");


    }
public static String procesarElemento (Node nodo)
  {
    String nombre= nodo.getNodeName();
    System.out.print("("+nombre);
    NamedNodeMap atributos= nodo.getAttributes();
    for (int i=0; i<atributos.getLength(); i++)
    { Attr at= (Attr)atributos.item(i);
      System.out.print(" "+at.getName()+"= "+at.getValue());
    }
    System.out.print(")");
    return nombre;
   }
}
```

Esto es una pequeña muestra de la potencia de DOM. Una descripción en mayor profundidad puede encontrarse en [Maruyama 00].

SAX

SAX [SAX] se diseñó como una API que no genera estructuras internas, en contraposición a DOM. Las aplicaciones deben registrar gestores de eventos para un objeto analizador que implemente la interfaz `org.sax.Parser`. SAX posee tres interfaces de gestores: `DocumentHandler`, `DTDHandler` y `ErrorHandler`. También incluye la clase de implementación por defecto `HandlerBase` para el funcionamiento

predeterminado de las anteriores interfaces. Podríamos decir, que SAX no es más que un analizador léxico tradicional [Aho 90] sobre el cual se induce el tratamiento del documento.

De todas las interfaces, `DocumentHandler` es la más importante, distinguiendo entre sus métodos:

Nombre del método	Descripción
`startDocument()`	Recibe notificación del principio del documento
`endDocument()`	Recibe notificación del final del documento
`startElement(String name, AttributeList atts)`	Recibe notificación del principio de un elemento
`endElement(String name)`	Recibe notificación del final de un elemento
`characters(char ch[], int start, int length`	Recibe notificación de los datos de caracteres

La versión SAX del programa DOM anterior es:

```
public class EjemploSAX extends HandlerBase {
    static public void main(String[] argv) {
      try {

        Parser parser=
        ParserFactory.makeParser("com.ibm.xml.parser.SAXDriver");
        IS imprimirSAX = new IS();
        parser.setDocumentHandler(imprimirSAX);
        parser.parse(argv[0]);

      } catch (Exception e) {
        e.printStackTrace();
      }
    }

    public void startDocument() throws SAXException {
      System.out.print("(documento)");
    }

    public void startElement(String nombre, AttributeList atributos) throws SAXException
{
      System.out.print("(" + nombre);
      for (int i = 0; i < atributos.getLength(); i++) {
        String nombreAt = atributos.getName(i);
        String valor = atributos.getValue(i);
        System.out.print(" "+nombreAt+"= "+valor);
      }
      System.out.print(")");
    }

    public void endElement(String nombre) throws SAXException {
      System.out.print("(/"+ nombre+")");
    }

    public void characters(char[] texto, int comienzo, int longitud) throws SAXException
{
      System.out.print(new String(texto, comienzo, longitud));
    }
}
```

Aunque en este ejemplo concreto parezca más sencilla la aproximación SAX que DOM, esto se debe a que no se requiere la construcción de un objeto concreto (por ejemplo de la clase `Empleado`) en base a la información de un elemento (por ejemplo `empleado`). Esto requiere listar en el método `startElement` todas los elementos XML posibles que requieren tratamiento (imaginemos un programa SAX que gestiona una DTD de cien elementos), el cual se produce junto con el método `endElement`. Como podemos ver el ahorro de la estructura interna, se traduce en un código más complejo de leer, desde nuestro punto de vista.

4.6 Conclusiones

En este capítulo hemos revisado distintos estándares relacionados con el dominio de los lenguajes de marcado. En lo referente a lenguajes para definir documentos marcados hemos podido comprobar un sesgo creciente en el uso de los lenguajes producidos. Así hemos pasado de un uso eminentemente dirigido al campo documental y editorial, a un uso propio de lenguajes informáticos clásicos, como puede ser [Shirota 01]. Con la moda existente en lenguajes de marcado no parece exagerado aventurar que si hoy se reformulase SQL, tendría una sintaxis de lenguaje de marcado (tampoco sería de extrañar que el trabajo desarrollado en el seno de SQLX [SQLX] llegase a un conclusión similar). Por tanto podemos afirmar que a partir del uso original de los lenguajes de marcado para estructurar documentos han ido apareciendo variantes, más o menos ortodoxas. Las que más se ajustan al concepto original son aquellas utilizadas para marcar estructuras de datos, o estructuras de algún tipo (p.e. navegacionales). Las más alejadas son aquellas que utilizan los lenguajes de marcas como lenguajes clásicos, derivando incluso en lenguajes de script. Si la piedra de toque en programación orientada a objetos para distinguir entre herencia y agregación es preguntar ¿la clase B es A, o la clase B pertenece a A?, la piedra de toque para distinguir la necesidad de aplicación de un lenguaje de marcas o uno tradicional puede ser: ¿a partir de los terminales del lenguaje se puede especificar un procesamiento basado en la sintaxis o no?. Si se puede, la opción más razonable (salvo cuestiones de legibilidad) es utilizar un lenguaje tradicional. Si no, debe utilizarse un lenguaje de marcas. A nuestro entender, este uso es excesivo e incluso nos hemos dejado influenciar por el en esta tesis. En efecto, el lenguaje propuesto para definir el esquema navegacional en esta tesis (Capítulo 6) está basado en sintaxis XML, cuando podría haberse utilizado una sintaxis tradicional. Nos hemos decantado a favor de XML por dos razones, que en última instancia justifican el auge de los lenguajes de marcas. La primera es la mayor simplicidad a la hora de construir una aplicación que trate dicho tipo de lenguajes, ya la disponibilidad de herramientas desarrolladas por terceras partes según diversos estándares nos ha evitado la construcción de un analizador clásico, bien descendente recursivo, o bien ascendente basado en programas YACC o similares. La segunda es disponer de un esquema navegacional en un formato de intercambio, que a través de transformaciones XSLT puede convertir en otros documentos como HTML o XUL, relacional [Bourret 01], u otro formato electrónico [Bryan 98], [Peat 97].

Respecto al estándar HyTime, verdadero esfuerzo en el campo hipermedia hemos podido comprobar como a pesar de no tener éxito en su aplicación si que tuvo bastante repercusión. Sus ideas han sido recogidas por una serie de estándares desarrollados con posterioridad. Así XPath y XPointer son similares al módulo de direcciones de localización, mientras que XLink es casi equivalente al módulo de hiperenlaces, aunque sin su formalización explícita. Incluso SMIL puede entenderse como una concreción del módulo de planificación. A nuestro entender el mayor problema de HyTime fue su complejidad. No solamente era necesario conocer el estándar, sino las facilidades extendidas que proporcionaba (al menos el tratamiento arquitectónico, y los groves). Además, no debemos olvidar, que al final de todo este proceso los documentos HyTime no tienen semántica de tratamiento predeterminada.

En cuanto a las aplicaciones de SGML/XML vemos como HTML, apoyado en su simplicidad, se muestra como el lenguaje de descripción hipermedia más extendido del mundo, a pesar de sus carencias. SMIL, por otro lado, a pesar de su potencia descriptiva no es capaz de capturar de manera sencilla y elegante una de las mayores necesidades en sistemas hipermedia: representar enlaces textuales. Esto no es una crítica al estándar, sino la constatación de su falta de idoneidad en un uso hipermedia extensivo de este lenguaje. Por último hemos visto dos lenguajes para descripción de interfaces de usuario. La mayor diferencia entre ambos es que uno se concibe para ser interpretado por un navegador, mientras que el otro genera código fuente para ser compilado. Al ser de propósito general deben cubrir un problema implícitamente resuelto en HTML y en SMIL: la gestión de eventos. XUL opta por utilizar código JavaScript (en la línea de ser interpretado por un navegador), mientras que UIML opta por describir los eventos para luego ligarlo a clases concretas (en la línea de producir código compilable). Desde el punto de vista hipermedia, y debido a su ejecución en un navegador, XUL es una opción más factible para representar este tipo de aplicaciones.

Finalmente hemos visto de forma muy breve una serie de estándares asociados a XML que facilitan el tratamiento de estos documentos. Esta es una de las mayores diferencias con respecto a SGML, que salvo DSSSL, no disponía de ningún lenguaje estándar asociado que facilitase su aplicación. De los vistos cabe destacar XPath, XLink, XSLT, y DOM por su uso en el generador de aplicaciones descrito en el Capítulo 6. El funcionamiento de este generador (descrito en el Apéndice B), basado por completo en el modelo hipermedia descrito en el Capitulo 5, puede resumirse en:

- Disponer de documentos XML que capturen los contenidos y relaciones entre estos (con XLink), con independencia del esquema navegacional, también descrito en XML.
- Relacionar ambos esquemas con relaciones XPath, o XSLT si fuera necesario.
- Realizar una aplicación Java que a través de un interfaz DOM:
  - Genere la interfaz de usuario (Swing) en base a la descripción del esquema navegacional.
  - Gracias a la relación de dicho esquema navegacional y en base a las expresiones XPath que lo relacionan con los contenidos, seleccionar los contenidos que van a aparecer en la aplicación, y mediante una transformación XSLT que se apoya en los enlaces XLink traducir los contenidos en formato XML a formato HTML.
  - Utilizar `JEditorPanels` SWING para mostrar las páginas HTML en la interfaz generada.
  - Proporcionar la semántica de navegación de la aplicación gracias a un controlador de eventos que implementa el interfaz `HyperlinkListener` Swing, que en combinación con el esquema navegacional muestra los contenidos en el lugar determinado.

Esta aproximación recoge la filosofía de tratamiento hipermedia con lenguajes de marcado que hemos visto, pero con ventajas. Respecto a HyTime proporciona una relación entre el nivel de contenidos y el de navegación, posibilitando así un tratamiento por defecto. Respecto a HTML gana estructuración, y permite independizar totalmente contenidos de esquema navegacional. Respecto a SMIL soluciona el problema de localizar anclas en contenidos textuales. Respecto a XUL y UIML evita el tener que proporcionar controladores de eventos explícitos para la gestión de enlaces. Evidentemente también presenta inconvenientes. Respecto a HyTime pierde genericidad al fijar la semántica de tratamiento. Respecto a HTML pierde universalidad de plataforma y sencillez. Respecto a SMIL presenta un tratamiento multimedia menos avanzado. Respecto a XUL y UIML pierde la capacidad de responder a otro tipo de eventos que no sea la selección de hiperenlaces. A pesar de estos inconvenientes pensamos que en nuestra aproximación son más importantes las ventajas que los inconvenientes ya que nuestro objetivo es construir de forma automática aplicaciones hipermedia, que van a funcionar como prototipos de la versión final.

Por último parece necesario hacer una reflexión sobre el "éxito comercial" de los estándares. ¿Por qué XML tiene el éxito que nunca tuvo SGML? ¿Mayor simplicidad? ¿Estándares más legibles? ¿Acceso gratuito a los mismos? ¿Mejor *marketing*? Quizás sea una pregunta que no corresponda contestar, en esta tesis. Aventurando una respuesta podríamos hablar de mejor *filosofía de venta* en XML que en SGML. Otra pregunta que cabe hacerse es ¿por qué si HTML es un lenguaje limitado que en diversas áreas fue sustituido por aproximaciones de mayor nivel como XML, nunca fue sustituido en el dominio hipermedia por HyTime? ¿Mayor simplicidad? ¿Mayor difusión? También aventuraremos una respuesta: *semántica.* A nuestro juicio HyTime hereda la mayor desventaja de la declaratividad SGML y del procesamiento arquitectónico, la falta de un tratamiento al documento marcado, y la necesidad de los procesadores arquitectónicos. En otras áreas, donde los documentos describen datos esto puede ser asumible, pero parece duro un tratamiento basado en groves como el descrito en la Figura 5 de este capítulo, donde aparte de un procesador arquitectónico y un motor HyTime, necesitamos una aplicación que procese el epGrove para generar la aplicación hipermedia. En contraposición HTML presenta un tratamiento tres veces más simple (navegador vs. motor arquitectónico + motor HyTime + aplicación HyTime).

5 El modelo Pipe

5.1 Introducción. Aplicaciones hipermedia estáticas y dinámicas

Hasta el momento no hemos proporcionado una definición concreta del término *hipermedia*. En este apartado analizaremos en detalle las connotaciones de este término y por tanto que aspectos debe ser capaz de caracterizar un sistema de representación hipermedia. También esbozaremos las diferencias (que serán formalmente definidas más adelante) entre una aplicación hipermedia *estática* y *dinámica*. Para empezar, retomemos la definición de aplicación hipermedia proporcionada por [Díaz 01]: "conjuntos de documentos multimedia organizados en una red hipertextual". Por documentos multimedia entendemos documentos que incluyen información textual, y no textual: imágenes, audio, vídeo, etc. Pero exactamente ¿cuál es la definición de hipertexto? Aunque numerosos autores describen la noción de hipertexto ninguno proporciona una definición concreta. Veamos distintas definiciones de hipertexto extraídas de diccionarios de diversa naturaleza técnica.

- El diccionario de *Maria Moliner* de uso del español [MM 98] define hipertexto como: "conjunto de convenciones que permite describir documentos, tanto su contenido (texto e imágenes) como su aspecto (márgenes, tipos de letras, etc.) e incluir enlaces con otros documentos". Como vemos es una definición demasiado apegada al concepto de HTML.
- El diccionario *Oxford English Dictionary* [DOC 96] lo define como: "texto que no forma una simple secuencia y que puede ser leído en varios órdenes; especialmente texto y gráficos, los cuales están interconectados de tal forma que un lector del material (por ejemplo mostrado en una pantalla de ordenador) puede interrumpir la lectura de un documento en ciertos puntos para consultar otro material relacionado". Esta definición define con mayor precisión el concepto de hipertexto..
- La *Enciclopedia de Computer Science* de *Nature* proporciona una definición más técnica [ECS 00]: "hipertexto es tanto el concepto de interrelacionar elementos de información (enlazar piezas de información) como el nombre utilizado para describir una colección o red de nodos enlazados o interrelacionados (un elemento de información o nodo puede varia desde una simple idea o trozo a un documento entero). Un sistema de hipertexto permite a un autor crear los nodos y los enlaces entre ellos, y permite al lector atravesar estos nodos, es decir, navegar de un nodo a otro utilizando estos enlaces". Esta última parece la más precisa y será la que tomemos en consideración.

Como podemos observar las definiciones anteriores se ajustan con mayor o menor fortuna a la noción de hipertexto que proporcionan implícitamente diversos autores, y que coincide con la imagen más habitual que tenemos todos nosotros. Entre distintos ejemplos de aplicaciones hipertexto podemos encontrar la propia recomendación de HTML 4.01, que hace el World Wide Web Consortium [W3C HTML]. Este ejemplo es casi exclusivamente hipertextual, y de gran utilidad para comprender el estándar. En contraposición, una aplicación eminentemente hipermedia es *Lire en Français* [Fernández-Valmayor 99], la cual utiliza con profusión contenidos multimedia y enlaces de hipertexto.

La recomendación W3C de HTML está disponible a través de Internet, proporcionando un gran ejemplo de las capacidades de difusión de esta valiosa herramienta. A pesar de este hecho cabe hacerse una pregunta ¿todos los documentos que se distribuyen en Internet (y en particular la World Wide Web) son documentos hipermedia? ¿Y las aplicaciones que se encarga de ejecutar el navegador son aplicaciones hipermedia? Parece razonable contestar que no, ya que un *chat*, por ejemplo, no parece ajustarse a al definición de hipertexto. Por tanto es necesario que los sistemas de representación hipermedia (y en especial aquellos utilizados para la fase de conceptualización, como veremos en el Capítulo 6 de esta tesis) definan con precisión que tipo de información quieren caracterizar, obviando todas aquellas características de una aplicación que no se vean incluidas en la definición de hipermedia tomada como referencia. Esto es lo que hacen la mayoría de sistemas de representación hipermedia analizados en el Capítulo 2. Solamente Labyrinth y OOHDM, en virtud a su capacidad de responder a eventos genéricos, son capaces de ir más allá de las características puramente hipermediales de una aplicación. El precio de esta posibilidad es la exclusión de una semántica de navegación por defecto, muy útil para la generación de prototipos de forma automática basados en estos sistemas de representación hipermedia [Nanard 98]. A otro nivel de abstracción podríamos decir que este es el problema que tienen aproximaciones genéricas como XUL o UIML, ya que al no determinar un tratamiento por defecto para la selección de enlaces deben proporcionarse explícitamente controladores de eventos para dicha selección.

En cuanto a la naturaleza estática o dinámica de una aplicación hipermedia hay que tener en cuenta si los enlaces y/o los contenidos pueden caracterizarse a priori, o necesariamente son caracterizados en tiempo de ejecución. Por ejemplo, si tenemos un texto en el que aparecen palabras que enlazan con su definición en un diccionario tenemos contenidos y enlaces estáticos (realmente esto puede complicarse, e incluso podría darse una caracterización dinámica. En la Figura 7.9 del Capítulo 7 puede encontrarse una discusión a este respecto). Por el contrario, si tenemos un formulario de búsqueda de tal forma que al hacer una pregunta se genera en tiempo de ejecución la definición (posiblemente con varias acepciones) que haya encontrado (supuesto que haya) tenemos contenidos y enlaces dinámicamente generados. Por tanto a las aplicaciones hipermedia cuyos contenidos y relaciones se establecen antes de la ejecución de la misma las denominaremos *aplicaciones hipermedia estáticas*. A las aplicaciones que establecen sus contenidos, y las relaciones entre estos en tiempo de ejecución las llamaremos *aplicaciones hipermedia dinámicas*. Volveremos más adelante sobre estos conceptos.

En este capítulo presentaremos el modelo hipermedia Pipe, pensado para caracterizar la etapa de conceptualización tanto de aplicaciones hipermedia estáticas como dinámicas (la Figura 6.2 del Capítulo 6 sitúa a Pipe dentro de un modelo de proceso). Pipe proporciona además una semántica de navegación para ambos tipos de aplicaciones, obviando una descripción en detalle de las actividades computacionales de propósito general. Esta descripción no se elude en el modelo de proceso, sino que se pospone hasta la etapa de diseño (entendida como la fase descrita en la Figura 6.2 del Capítulo 6), utilizando para su caracterización, por ejemplo, un mecanismo de modelado de propósito general como UML. Los sistemas de representación hipermedia con semántica de presentación por defecto que analizamos en el Capítulo 2 no eran capaces de caracterizar contenidos dinámicos. La solución que proporciona Pipe para conciliar la semántica de navegación con la existencia de contenidos y enlaces dinámicos, es la caracterización de los enlaces establecidos entre los contenidos mediante una función, *la función de relación*. Cuando la imagen de esta función pueda determinarse antes de ejecutar la aplicación tendremos aplicaciones hipermedia estáticas. En otro caso serán dinámicas.

Además Pipe parte de una escrupulosa separación del nivel de contenidos y del esquema presentacional que se va a proporcionar a estos. Para ello utiliza dos funciones. La primera asigna contenidos (estáticos o dinámicos) a los elementos del esquema navegacional. La segunda asigna enlaces de contenidos (estáticos o dinámicos) a enlaces establecidos entre los elementos del esquema navegacional, los cuales denominaremos *tuberías*. De esta forma diremos que las relaciones establecidas a nivel de contenidos *fluyen* por las tuberías establecidas a nivel presentacional.

Pipe también soporta la noción de proceso subordinado. Dichos procesos son pequeñas aplicaciones que se ejecutan dentro de la aplicación hipermedia, pero que en ningún caso pueden afectar a la navegación en la misma. Otra vez Pipe elude una descripción detallada de dichos procesos, quedando relegada esta tarea para la etapa de diseño. Además, Pipe no se preocupa de modelar ninguna interacción más allá que la correspondiente a la selección de un enlace, interacción básica y fundamental en una aplicación hipermedia.

Por último Pipe también soporta nociones genéricas dentro del contexto hipermedia como la sincronización, o la noción de contexto, no soportada por todos los sistemas de representación hipermedia analizados previamente.

A continuación veremos el modelo Pipe en profundidad.

5.2 Pipe

La representación de una aplicación hipermedia a través del modelo Pipe [Navarro 02] es una tupla <GC, EN, FC, SN, SP> donde:

- GC: es el *grafo de contenidos*. Es la parte encargada de modelar los contenidos y los enlaces entre estos, con independencia de la interpretación navegacional que se vaya a dar a ambos.
- EN: es el *esquema navegacional*. Es la parte encargada de modelar los elementos de la interfaz de usuario de la aplicación y las relaciones navegacionales establecidas entre estos.
- FC: son las *funciones de canalización*. Es la parte encargada de interpretar navegacionalmente el grafo de contenidos a través del esquema navegacional.

- SN: es la *semántica navegacional*. Es la encargada de definir que contenidos se van a presentar en el esquema navegacional como resultado de seleccionar un enlace navegacional que tiene asignado (canaliza) un enlace del nivel de contenidos. En palabras de [Stotts 89], es la forma en que va a ser accedida y presentada la información.
- SP: es la *semántica presentacional*. Es la semántica navegacional, más una función que informa de las características presentacionales concretas (fuente, color, tamaño, etc.) que se aplica a los contenidos.

Para caracterizar una aplicación hipermedia será necesario proporcionar el grafo de contenidos, el esquema navegacional, y las funciones de relación en cada caso, mientras que las semánticas de navegación y presentación son comunes para todas las aplicaciones. Veamos en detalle cada componente del modelo.

5.2.1 Grafo de contenidos

5.2.1.1 Contenidos

Como su nombre indica, el *grafo de contenidos* de Pipe, *GC*, es un grafo que sirve para modelar los contenidos de la aplicación. Esta aproximación coincide con la propuesta por la máquina abstracta de hipertexto pero ampliada para dar cabida a la generación dinámica de contenidos y enlaces por parte de la aplicación hipermedia. Es decir, los nodos del grafo representan los contenidos de la aplicación, mientras que los arcos del grafo representan los enlaces existentes entre estos. Dicho de otra forma, el grafo de contenidos caracteriza el nivel de almacenamiento del modelo Dexter.

Definimos C como el *conjunto de contenidos susceptibles de aparecer en una aplicación hipermedia*. Dividimos este conjunto en dos,

$$C = C_{ns} \cup C_s, \; C_{ns} \cap C_s = \varnothing$$

Donde C_{ns} es el conjunto de *contenidos no subordinados*, y C_s es *el conjunto de contenidos subordinados*. Veamos en que se diferencian. Los contenidos no subordinados representan los contenidos de la aplicación hipermedia que se muestran al usuario, en su acepción del modelo Dexter: texto, vídeo, audio, etc. Pipe no da importancia a la naturaleza interna de los contenidos. Es decir, desde el punto de vista de este modelo es exactamente igual un mapa del mundo en el que aparecen referencias a países, que un texto sobre el mundo en el que aparecen referencias a países. La razón de este hecho se debe a que Pipe intenta ser lo más genérico posible y, por tanto, independiente de técnicas concretas de implementación. En ambos casos su asimilación al modelo se producirá a través de un nombre (identificador) que le corresponderá en el conjunto C. Por otro lado, esta técnica es bastante similar a la utilizada por la mayoría de sistemas de representación hipermedia analizados en el Capítulo 2.

Lo importante desde el punto de vista del modelo es que dicho contenido represente un dato (en cuyo caso puede ser origen o destino de enlaces) y pertenezca al conjunto C_{ns}, o represente un proceso (en cuyo caso a lo sumo solo puede ser destino de un enlace) y pertenezca al conjunto C_s. Es por esta razón que en los contenidos distinguimos a los contenidos subordinados, los cuales representan a los *procesos subordinados*. Estos son pequeñas aplicaciones que se ejecutan dentro de la aplicación hipermedia, pero que no afectan a la navegación en la misma. Una calculadora dentro de un panel de una ventana, es un posible ejemplo de dichas aplicaciones. Aunque podíamos haber optado por evitar una caracterización dicotómica de los contenidos (tal y como hace el modelo Trellis que no distingue entre datos y procesos) hemos preferido una separación explícita entre estos, ya que permite una mayor claridad a la hora de proporcionar una representación de una aplicación hipermedia. Además, como veremos a continuación, los contenidos subordinados no pueden ser el origen de ningún hiperenlace, siendo esta la razón principal de la segregación respecto de los contenidos no subordinados.

Nótese que los contenidos pueden tener otro tipo de relaciones entre ellos (por ejemplo agregación), o pueden ajustarse a las normativas de diversos estándares (como por ejemplo a los distintos estándares de *e-learning*) el modelo obvia tales características, ya que no van a ser relevantes para el mismo (aunque por supuesto si que lo sean en las fases de diseño y construcción). De la misma forma, los procesos subordinados pueden entenderse como métodos de objetos, o como funciones independientes. Al igual que con los contenidos esas características no son del interés del modelo.

Si queremos caracterizar enlaces entre contenidos tenemos la necesidad de especificar la existencia de anclas dentro de los mismos. Con ese fin introducimos el *conjunto de anclas*, A, en su acepción del modelo Dexter (es decir, posiciones en los contenidos que denotan el origen o destino de un enlace). Este conjunto debe ser capaz de hacer frente a la tarea de localizar posiciones dentro de contenidos de naturaleza sumamente heterogénea: una imagen GIF, un documento XML, un vídeo AVI, etc. Por tanto Pipe, al igual que con los contenidos, no se preocupa de la naturaleza concreta del conjunto A. Como solución genérica A puede considerarse como un subconjunto del conjunto de cadenas de caracteres. De esta forma la expresión (12, 3) puede denotar un ancla que se localiza en el carácter 12 de un texto y tiene una extensión de 3 caracteres. De la misma forma la expresión (1, 1, 10, 5) puede denotar un rectángulo dentro de una foto (en coordenadas de la misma, por ejemplo). Dentro del modelo esta distinción no tiene demasiada importancia, ya que las anclas se utilizan fundamentalmente con el fin de identificar enlaces entre contenidos, y al igual que con los contenidos, son asimiladas al modelo por un nombre en el conjunto A. En este nivel se mantiene este grado de generalidad porque, tal y como veremos en el Capítulo 6 de esta tesis, el modelo Pipe se concibe como una herramienta de conceptualización, y no de diseño. Evidentemente técnicas concretas de prototipado y/o construcción de aplicaciones basadas en este modelo (como la presentada en el Capítulo 6) necesitarán representaciones concretas de los contenidos y de sus anclas.

El conjunto A denota a todas las anclas susceptibles de aparecer en un contenido de una aplicación hipermedia.

Para relacionar a los contenidos con sus anclas necesitamos definir la *función de anclaje* a:

$$a: C \rightarrow \wp(A)$$

tal que a un contenido le asigna el conjunto de anclas, tanto origen como destino, que tiene asociado (en la definición anterior, y a lo largo de esta tesis, el conjunto $\wp(X)$ denota el conjunto partes de X). En el caso de contenidos no subordinados estas anclas podrán ser tanto origen como destino de enlaces. En el caso de contenidos subordinados estas anclas solo pueden corresponderse con anclas interpretadas como destino. En particular solo podrá tomar el valor total, que denota la totalidad de un contenidos (subordinado o no). Como veremos más adelante, en Pipe no existen *anclas origen* o *anclas destino*, ya que esta no es una característica innata del ancla, sino adquirida en la definición de enlace.

Además podemos suponer que:

$$\forall c \in C, \text{total} \in a(c)$$

donde total es un ancla que denota la posición inicial en cualquier contenido.

5.2.1.2 Enlaces y relaciones estáticos

Enlaces estáticos

Ahora nos encontramos en disposición de definir el *conjunto de enlaces de la aplicación hipermedia estática*, E^0, como:

$$E^0 \subseteq C_{ns}{}^0 \times A^0 \times C^0 \times A^0$$

Donde:

- $C^0 \subseteq C$ es el *conjunto de contenidos estáticos de una aplicación hipermedia*, con $C^0 = C_{ns}{}^0 \cup C_s{}^0$, $C_{ns}{}^0 \subseteq C_{ns}$, $C_s{}^0 \subseteq C_s$
- $A^0 \subseteq A$ es el *conjunto de anclas de una aplicación hipermedia estática*. Nótese que, $A^0 = \bigcup_{c \in C^0} a(c)$

Veamos que denota un enlace e = (c, a, c', a') perteneciente al conjunto anterior. La primera coordenada, c, denota el contenido donde se origina el enlace. La segunda coordenada, a, denota el ancla donde se origina el enlace. La tercera coordenada, c', denota el contenido destino del enlace. Finalmente la cuarta coordenada, a', denota el ancla en el contenido destino. Nótese que para representar un enlace n-ario es necesario repetir la información n veces. Es decir, si el contenido a enlaza con el b y el c necesitamos en principio los enlaces (a, aBC, b, total), (a, aBC, c, total). Esta idea básica presenta ciertas complicaciones, y aunque la formulación del

modelo aquí propuesto no cubre en su totalidad la posibilidad de enlaces n-arios dinámicos, en el Apéndice A se presenta una formulación que si la cubre.

En Pipe el papel de origen o destino de un ancla viene dado por el propio enlace y no por el ancla. Esta aproximación, a otro nivel de abstracción, es similar a la propuesta por HyTime con sus enlaces `hylink` o por XLink con sus enlaces `extended`., tal y como vimos en el Capítulo 4. En el contexto de los sistemas de representación hipermedia, aproximaciones como [Díaz 01] también optan por esta solución.

Además Pipe solo considera enlaces unidireccionales, ya que la existencia de anclas imposibilita, al menos desde el punto de vista de consistencia semántica, la bidireccionalidad del enlace. En efecto, supongamos que una biografía enlaza con los países visitados por el sujeto de la biografía, y en particular permite enlazar con los países agrupados por continentes. Supongamos también que uno de estos continentes es Europa. En este caso deberíamos tener un enlace de la forma (biografía, aEuropa, países, europeos). Si el enlace fuese bidireccional, en enlace de la lista de países con la biografía partiría de la zona de países europeos, y tendría como destino el origen del enlace, cosa bastante poco deseable, ya que la relación entre países y biografía, no tiene origen en los países europeos exclusivamente. La Figura 5.1.a ilustra esta situación. Si se desea indicar la existencia de un enlace entre la lista de países y la biografía debería incluirse un enlace de la forma (países, aBiografía, biografía, total), donde el ancla aBiografía permite seleccionar una posición concreta dentro del contenido países, y no en particular la correspondiente a países europeos. La Figura 5.1.b ilustra esta situación.

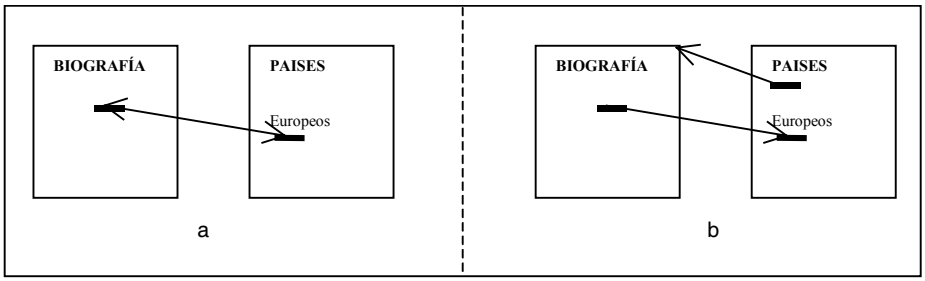

Figura 5.1 En el caso a, el ancla origen del enlace entre PAISES y BIOGRAFÍA coincide con el ancla destino del enlace entre BIOGRAFÍA y PAISES. En el caso b, el ancla origen del enlace entre PAISES y BIOGRAFÍA es independiente del ancla destino entre BIOGRAFÍA y PAISES. Las pequeñas marcas de los contenidos denotan las anclas. El perímetro exterior del contenido denota al ancla total.

Nótese que la definición de enlace proporcionada no excluye la posibilidad de agrupar enlaces del mismo tipo (es decir, que relacionan al mismo tipo de entidades). Esta aproximación lleva a una solución similar a la propuesta por los sistemas de representación hipermedia relacionales (HDM y RMDM). En el apartado 5.4 se discute esta posibilidad, aunque en la práctica decide no aplicarse en aras de una mayor generalidad y potencia expresiva.

Veamos a continuación un ejemplo de enlaces entre contenidos. Con este fin retomaremos el ejemplo visto en el capítulo de sistemas de representación hipermedia. Empezaremos restringiéndonos a la parte estática. En dichos contenidos aparece un texto inicial que contiene una breve biografía del profesor. Esta biografía enlaza con un mapa España, país de nacimiento del autor, y viceversa (es decir, el mapa de España también presenta un enlace con la biografía). Dentro de este mapa tenemos enlaces con descripciones de diversas ciudades de la geografía española que son de interés para el profesor, y viceversa. En particular destacan las ciudades de Madrid, y la localidad de Lepe. La biografía también presenta un enlace a una descripción los estudios cursados por el profesor, y viceversa. También enlaza con una descripción de la docencia que imparte el profesor y viceversa. Además, la biografía permite enlazar con descripciones concretas a cada asignatura dentro de la descripción genérica de la docencia. La descripción de asignaturas enlaza con el programa de cada una de ellas y viceversa. Además hay un índice representado por una imagen que permite acceder al país de nacimiento, a los estudios y a la docencia. La Figura 5.2 representa gráficamente esta información.

Su formato conjuntista es:

C^0 = { biografía, índice, españa, madrid, lepe, estudios, docencia, isg1, isg2, isg91 }

E^0 = { (biografía, aEsp, españa, total), (españa, aBio, biografía, total),
(españa, aMad, madrid, total), (madrid, aEsp, españa, total), (españa, aLepe, lepe, total),
(lepe, aEsp, españa, total),
(biografía, aEst, estudios, total), (estudios, aBio, biografía, total),
(biografía, aDoc, docencia, total), (docencia, aBio, biografía, total),
(biografía, aDocIsg1, docencia, isg1), (biografía, aDocIsg2, docencia, isg2),
(biografía, aDocIsg91, docencia, isg91),
(docencia, aIsg1, isg1, total), (isg1, aDoc, docencia, total),
(docencia, aIsg2, isg2, total), (isg2, aDoc, docencia, total),
(docencia, aIsg91, isg91, total), (isg91, aDoc, docencia, total),
(índice, aEsp, españa, total), (índice, aEst, estudios, total), (índice, aDoc, docencia, total) }

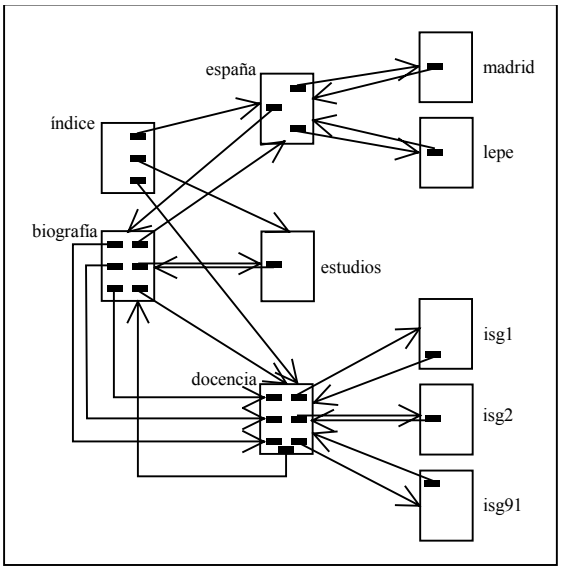

Figura 5.2 Grafo de contenidos de la aplicación.

En este ejemplo se puede apreciar como los contenidos y las anclas no son más que identificadores, siendo los contenidos y anclas reales de una naturaleza heterogénea de la cual no se preocupa el modelo. Nótese además que en el conjunto E^0 hay cuatro enlaces entre biografía y docencia uno que da acceso a toda la docencia, y tres que se centran en las partes correspondientes a las asignaturas.

La función de relación

La definición del conjunto E^0 proporcionada en el apartado anterior es demasiado genérica. En última instancia el conjunto E^0 debe denotar una relación binaria entre pares de contenido origen y ancla origen con contenido destino y ancla destino. Para una caracterización más elegante de la semántica navegacional de la aplicación, y para (como veremos en el siguiente apartado) permitir la presencia de contenidos dinámicos, vamos a dividir los enlaces en su parte origen (contenido y ancla) y en su parte destino (contenido y ancla). Con este fin, y para caracterizar la relación inducida entre contenidos y anclas en base a enlaces, definimos la *función de relación estática*, r,

$$r: C_{ns}^{\ 0} \times A^0 \rightarrow C^0 \times A^0$$

tal que si $(c, a) \in C_{ns}^{\ 0} \times A^0$, r(c, a) es el contenido (y su ancla) con el que se relaciona el contenido c a través del ancla a. La función r no es total, ya que en el conjunto $C_{ns}^{\ 0} \times A^0$ hay pares (c, a) que no van a ser el origen de ningún enlace. Además, no es inyectiva, ya que es posible que dos contenidos anclados distintos lleven al mismo destino. Finalmente no es suprayectiva, ya que la imagen está contenida en un conjunto demasiado genérico.

La función de relación r nos va a permitir caracterizar los enlaces en base a una función, independizando de esta forma el origen del destino. En el caso de que los enlaces sean estáticos, la función r simplemente va a relacionar contenidos y anclas con contenidos y anclas. En el caso de que existan contenidos dinámicos, la función r estará parcialmente definida en base a una función de generación de contenidos (veremos esto en el

siguiente subapartado). Por lo tanto la función r actúa como una caja negra que permite caracterizar de manera uniforme los enlaces entre contenidos, con independencia de que estos sean estáticos o dinámicos. En efecto, en términos de la función r podemos caracterizar al conjunto E^0, de la siguiente manera:

$$E^0 = \{ (c, a, r(c, a)) \mid (c, a) \in C_{ns}{}^0 \times A^0 \}$$

Por lo tanto para describir el conjunto de enlaces de una aplicación hipermedia no tenemos más que proporcionar la definición de la función r. Cuando la aplicación sea totalmente estática podremos definir a r por extensión, es decir, proporcionando el conjunto E^0. Cuando tenga componentes dinámicas, la definición de r deberá producirse de manera intensional, tal y como veremos más adelante en el apartado 5.2.1.3.

La Tabla 5.1 recoge al conjunto de enlaces del ejemplo desde esta perspectiva.

r: $C_{ns}{}^0 \times A^0$	$C^0 \times A^0$
(biografía, aEsp)	(españa, total)
(españa, aBio)	(biografía, total
(españa, aMad)	(madrid, total)
(madrid, aEsp)	(españa, total)
(españa, aLepe)	(lepe, total)
(lepe, aEsp)	(españa, total)
(biografía, aEst)	(estudios, total)
(estudios, aBio)	(biografía, total)
(biografía, aDoc)	(docencia, total)
(docencia, aBio)	(biografía, total)
(biografía, aDocIsg1)	(docencia, isg1)
(biografía, aDocIsg2)	(docencia, isg2)
(biografía, aDocIsg91)	(docencia, isg91)
(docencia, aIsg1)	(isg1, total)
(isg1, aDoc)	(docencia, total)
(docencia, aIsg2)	(isg2, total)
(isg2, aDoc)	(docencia, total)
(docencia, aIsg91)	(isg91, total)
(isg91, aDoc)	(docencia, total)
(índice, aEsp)	(españa, total)
(índice, aEst)	(estudios, total)
(índice, aDoc)	(docencia, total)

Tabla 5.1 Caracterización del conjunto de enlaces E^0 a través de la función de relación r

Contenido accedido y enlace activado

A continuación introduciremos dos funciones que nos van a ayudar a definir la semántica de navegación. La primera, conAcc, sirve para caracterizar al contenido accedido a través de un enlace como consecuencia de la selección de un ancla. La segunda, enlAct, sirve para caracterizar al propio enlace activado. En ambos casos, las funciones se definen en base a la función de relación r.

En efecto, definimos

$$\begin{aligned} \text{conAcc: } C_{ns}{}^0 \times A^0 &\rightarrow C^0 \\ (c, a) &\rightarrow \prod_1 r(c, a) \end{aligned}$$

Donde la función $\prod_i$ representa la proyección de la coordenada i-ésima de una tupla. Esta función caracteriza los contenidos accedidos ($\prod_1 r(c, a)$) a través de un enlace ((c, a, r(c, a))) como consecuencia de la selección de un ancla (a) en un contenido (c). Nótese que esta función se encarga de caracterizar la composición de las funciones de resolución y acceso del modelo Dexter, siendo en este caso la función de resolución la identidad, y la de acceso la propia función conAcc. Es decir, si:

resolucion: Especificaciones → IDs
acceso: IDs → Componentes

Entonces conAcc = acceso∘resolución

También definimos la función:

$$\text{enlAct: } C_{ns}^{0} \times A^{0} \rightarrow E^{0}$$
$$(c, a) \rightarrow (c, a, r(c, a))$$

Esta función caracteriza al enlace activado ((c, a, r(c, a))) como consecuencia de la selección de un ancla (a) en un contenido (c). Aunque por ahora no aplicaremos estas funciones, su definición nos será de una utilidad extrema a la hora de definir la semántica navegacional de la aplicación hipermedia. La Figura 5.3 ilustra estos conceptos.

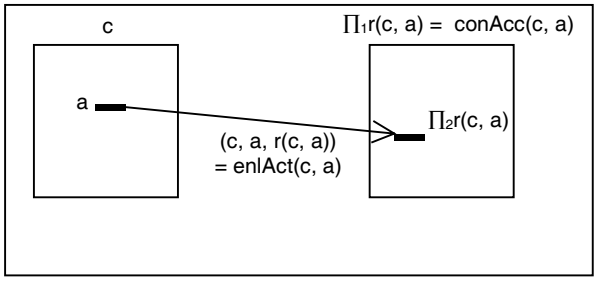

Figura 5.3 Contenido accedido y enlace activado.

Ninguna de estas funciones es total, ya que sus dominios son demasiado genéricos. En principio la semántica de navegación actúa solamente en la parte del dominio donde están definidas ambas funciones. De no ser así, se tendrían enlaces colgantes (*dangling links* [Hugh 99]), es decir, enlaces con origen pero sin destino. En cualquier caso de manera casi trivial podrían convertirse en totales considerando un contenido error con el que enlazan aquellos contenidos anclados sin imagen por la función de relación r.

Nótese que el modelo tampoco se preocupa de las posibles estructuras jerárquicas que pudieran derivarse de la organización hipertextual. En [Fernández 98a], [Fernández 98c], y [Fernández 98e] pueden encontrarse criterios de organización del conocimiento aplicables a este dominio.

5.2.1.3 Enlaces y relaciones dinámicos

La función de generación

La ventaja de caracterizar los enlaces a través de una función de relación r radica en la posibilidad de representar aplicaciones dinámicas. Cuando los contenidos, y enlaces entre estos, sean conocidos a priori, es decir, antes de la ejecución de la aplicación nos encontraremos con *aplicaciones hipermedia estáticas* (las vistas hasta ahora). En este caso podremos definir el conjunto de enlaces enumerando por extensión a la función de relación r (es decir, enumerando el conjunto E^0). Por el contrario, cuando sea la aplicación hipermedia la encargada de generar estos contenidos (por ejemplo, mediante una consulta a una base de datos), y los enlaces entre estos, nos encontraremos ante *aplicaciones hipermedia dinámicas*. En este caso para definir el conjunto de enlaces proporcionaremos una descripción intensional de la función de relación r en base a la que denominaremos *función de generación*. Por lo tanto, en ambos casos (estático o dinámico) la definición del conjunto de enlaces se lleva a cabo exclusivamente sobre la función de relación r. Como la semántica de navegación está definida sobre la función de relación r, la naturaleza estática o dinámica de la aplicación hipermedia es transparente para dicha semántica navegacional. De aquí la necesidad de definir la función de relación r.

La función de generación se va a encargar de considerar todos los posibles contenidos generados a partir de un contenido y un ancla. Definimos *función de generación* g:

$$g: C_{ns} \times A \rightarrow \wp(C \times A)$$

El rango de esta función se considera en $\wp(C \times A)$ ya que es básicamente no determinista, y dicho conjunto sirve para poder caracterizarla en la práctica.

Extendemos la función de relación r de una aplicación hipermedia como:

$$r: C_{ns} \times A \rightarrow \wp(C \times A)$$
$$(c, a) \rightarrow \{ (c', a') \}, \text{ si } (c, a, c', a') \in E^0$$
$$(c, a) \rightarrow g(c, a), \text{ eoc}$$

Es decir, para caracterizar una aplicación hipermedia vamos a considerar los contenidos y enlaces estáticos originales, y a partir de algunos de estos contenidos originales generaremos los contenidos y enlaces dinámicos de la aplicación. La función r se encargará en ambos casos de proporcionar el contenido y ancla con los que se relacionan un contenidos y ancla origen a través de un enlace (bien estático o bien dinámico). Al igual que g, la función r es parcial, no inyectiva y no suprayectiva.

Si llamamos E al conjunto de todos los enlaces posibles, es decir,

$$E = C_{ns} \times A \times C \times A$$

ahora conAcc y enlAct denotan al contenido accedido y enlace activado en un cómputo determinado:

$$conAcc: C_{ns} \times A \rightarrow C$$
$$(c, a) \rightarrow c', \text{ tal que } c' \in \prod_1 r(c, a)$$

$$enlAct: C_{ns} \times A \rightarrow E$$
$$(c, a) \rightarrow (c, a, c', a'), \text{ tal que } (c', a') \in r(c, a)$$

Como podemos ver, la función de relación r, hace como verdadera caja negra que permite obviar la naturaleza estática o dinámico de los enlaces y/o relaciones. La Figura 5.4 ilustra esta situación en el caso de un contenido dinámico.

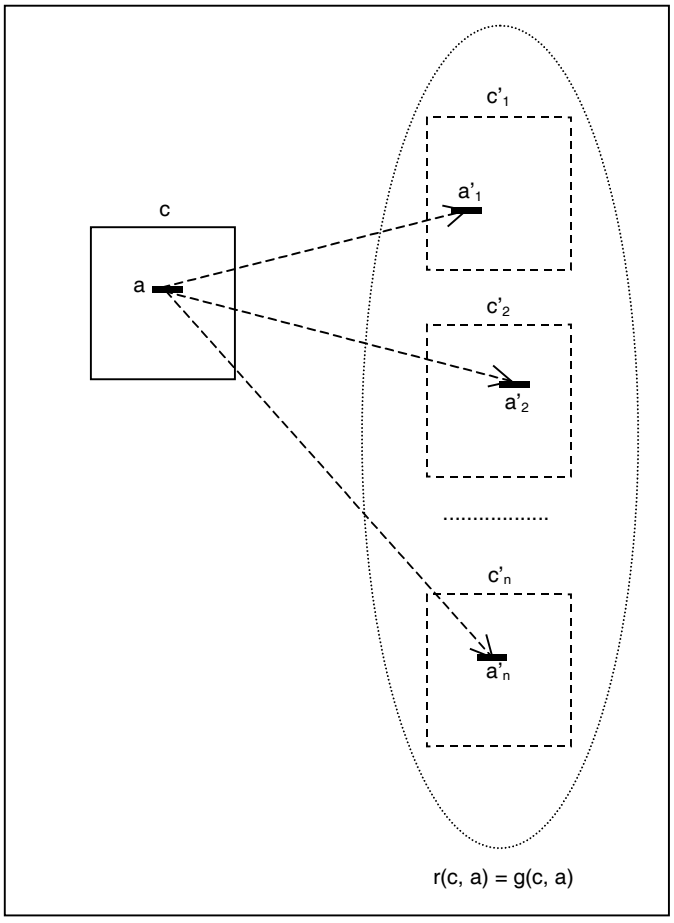

Figura 5.4 Generación dinámica de contenidos y enlaces

5.2.1.4 Aplicación de Pipe

Vamos a ver ahora una serie de conceptos que son necesarios para aplicar Pipe en casos prácticos.

Contenido generados, enlace generados, y contenido anclados

A continuación introduciremos unas funciones que no representan más que azúcar sintáctico para distintas vistas de la función de generación g. Dichas funciones no introducen ningún concepto nuevo y simplemente se encargan de proyectar convenientemente a la función g para su aplicación.

Definimos la *función de contenidos generados*, cg,

$$\begin{aligned} cg:\ & C_{ns} \times A \rightarrow \wp(C) \\ & (c, a) \rightarrow \prod_1 g(c, a) \end{aligned}$$

Esta función se encarga de caracterizar al contenido generado como consecuencia de la activación de un ancla (a) en un contenido (c).

La segunda función es la *función de enlaces generados*, eg,

$$\begin{aligned} eg:\ & C_{ns} \times A \rightarrow \wp(E) \\ & (c, a) \rightarrow \{ (c, a, c', a') \mid (c', a') \in g(c, a) \} \end{aligned}$$

Aun necesitamos una tercera función que nos será útil para aplicar el modelo en casos prácticos, la *función de contenido anclado*, ca,

$$\begin{aligned} ca:\ & \wp(C) \rightarrow \wp(C \times A) \\ & C' \rightarrow \cup_{c' \in C'} \{ (c', a') \mid a' \in ACO(c') \} \end{aligned}$$

donde la función ACO: $C \rightarrow \wp(A)$ se encarga de caracterizar las anclas consideradas como origen de nuevos enlaces que se encuentran en un contenido generado dinámicamente.

Esta función se encarga de relacionar las anclas generadas consideradas como origen en un contenido generado con dicho contenido. Su utilidad es meramente práctica. La Figura 5.5 ilustra este concepto supuesto que actúa sobre un elemento en la imagen de la función g.

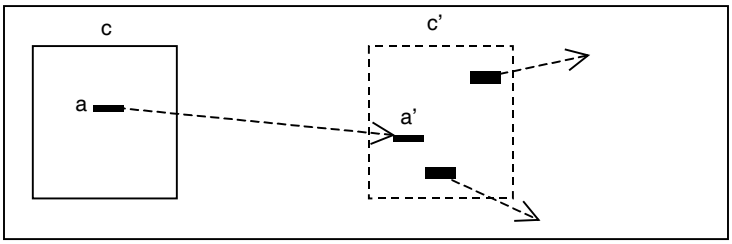

Figura 5.5 Las anclas de mayor grosor son las que caracteriza la función de contenidos anclados, y son precisamente aquellas que pertenecen al conjunto ACO(c'), es decir, las que serán origen de un enlace. Nos interesa esta caracterización para poder aplicar el modelo en la práctica

Contenidos generados, enlaces generados

En la práctica dos de las tres funciones anteriores se aplican tanto a elementos como a conjuntos de elementos, por esta razón vamos a extender su definición a conjuntos. En efecto, sea $CA \subseteq C_{ns} \times A$ definimos,

La *función de contenidos generados*, cg,

$$\begin{aligned} cg:\ & \wp(C_{ns} \times A) \rightarrow \wp(C) \\ & CA \rightarrow \cup_{(c, a) \in CA} cg(c, a) \end{aligned}$$

La segunda función es la *función de enlaces generados*, eg,

$$\begin{aligned} eg:\ & \wp(C_{ns} \times A) \rightarrow \wp(E) \\ & CA \ \rightarrow \cup_{(c,\, a)\in CA} eg(c, a) \end{aligned}$$

Contenidos y enlaces de aplicaciones dinámicas

Ahora estamos en condiciones de definir cuales son los contenidos de la aplicación hipermedia y sus enlaces en base a la función de generación g. De esta forma si hay $N \geq 1$ aplicaciones de la función de generación tenemos que, los contenidos de la aplicación C_A, y los enlaces de la misma E_A son:

$$C_A = C^N$$
$$E_A = E^N$$

siendo:

$$C^k = C^{k-1} \cup cg(CA^{k-1})$$
$$E^k = E^{k-1} \cup eg(CA^{k-1})$$
$$CA^k = ca(cg(CA^{k-1}))$$

para $1 \leq k \leq N$, C^0 *los contenidos estáticos*, E^0 los *enlaces estáticos*, y CA^0 los *contenidos anclados iniciales*. Estos contenidos anclados iniciales son los contenidos que iniciarán la primera aplicación de la función de generación g. Es decir, los contenidos y enlaces de la aplicación son los estáticos más los generados en sucesivas iteraciones de la función de generación g.

Además, debemos exigir una consistencia en las anclas de los contenidos ya existentes, respecto a los que se generan. Por tanto debemos indicar que:

$$\forall (c, a)(c' \in (cg(c, a) \cap C^0) \Rightarrow ACO(c') = \{\, a' \mid (c', a') \in CA^0 \,\})$$

Esta restricción indica que si generamos un contenido estático no podemos modificar las anclas que sirven para generar contenidos dinámicos (ni ninguna otra).

También debemos indicar que no podemos generar el mismo contenido dinámico dos veces, es decir:

$$\forall k\ 1 \leq k \leq N\ ((cg(CA^{k-1}) \cap (C^{k-1} \setminus C^0)) = \varnothing)$$

También debemos exigir que si generamos un contenido subordinado este no puede contener anclas consideradas como origen.

$$\forall (c', a') \in g(c, a)(c' \in C_s \Rightarrow ACO(c') = \varnothing)$$

Como la función de generación se aplica N veces tenemos que:

$$\forall c' \in C^N (ACO(c') = \varnothing)$$

Ejemplo

Veamos cual es la idea que subyace tras la función g a través de un ejemplo, pero antes ampliemos el que venimos considerando, suponiendo que tanto el índice como la biografía, enlazan con un formulario de búsqueda. Al igual que con el resto de contenidos obviamos una caracterización en detalle de la estructura de dicho formulario. Sabemos que es un contenido y que tiene el ancla botón. Por otro lado, a este nivel, es la única información que deseamos considerar. Dicho formulario generará una hoja de resultados con enlaces a distintas contenidos de la aplicación cuando sea seleccionada su ancla botón. Por tanto, si en el ejemplo de la Figura 5.2 tomamos como contenidos iniciales al conjunto:

C^0 = { biografía, índice, españa, madrid, lepe, estudios, docencia, isg1, isg2, isg91 } ∪ { formulario }

y tomamos como conjunto de enlaces estáticos E^0 a los enlaces definidos en la Tabla 5.1 junto con los enlaces (índice, aForm, formulario, total), (biografía, aForm, formulario, total), nos encontramos en una situación como la descrita por la Figura 5.6.

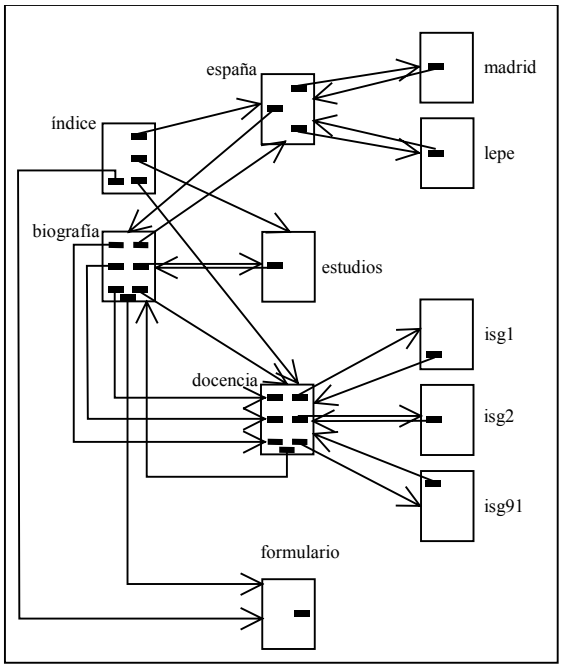

Figura 5.6 Contenidos y enlaces estáticos de la aplicación

Si ahora consideramos como conjunto de contenidos anclados iniciales al conjunto CA^0 = { (formulario, botón) }. Tenemos que una primera aplicación de la función de generación produce los resultados de la Tabla 5.2.

k=1 g:

CA^0	$P(C \times A)$
(formulario, botón)	g(formulario, botón) = {(resultado$_1$, total), (resultado$_2$, total), ..., (resultado$_n$, total) }

Tabla 5.2. Primera aplicación de la función de generación g

Según esto, los contenidos y enlaces generados por la función de generación al aplicarla sobre los contenidos y enlaces estáticos iniciales son:

$C^1 = C^0 \cup cg(\text{formulario, botón}) = C^0 \cup \{ \text{resultado}_1, \text{resultado}_2, ..., \text{resultado}_n \}$
$E^1 = E^0 \cup eg(\text{formulario, botón}) = E^0 \cup \{ (\text{formulario, botón, resultado}_1, \text{total}), ..., (\text{formulario, botón, resultado}_n, \text{total}) \}$
$CA^1 = ca(cg(\text{formulario, botón})) = ca(\{\text{resultado}_1, \text{resultado}_2, ..., \text{resultado}_n \}) = \{(\text{resultado}_1, a_1), (\text{resultado}_1, a_2), ..., (\text{resultado}_i, a_j), ..., (\text{resultado}_n, a_m)\}$

Esta situación queda reflejada en la Figura 5.7.

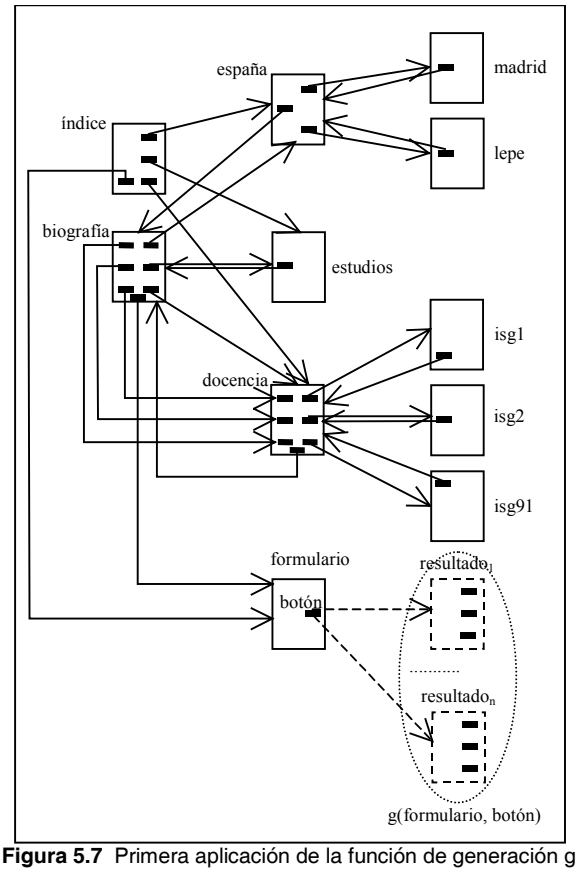

Figura 5.7 Primera aplicación de la función de generación g

Nótese que los contenidos y enlaces dinámicos están en línea discontinua. A partir de ahora los rectángulos de trazo sólido representarán contenidos estáticos. Los rectángulos de trazo discontinuo contenidos dinámicos. Los arcos continuos representan enlaces cuyo destino es un contenido estático. Los arcos discontinuos representan enlaces cuyo destino es un contenido dinámico. La Figura 5.8 proporciona una representación gráfica de los diferentes tipos de nodos y enlaces que podemos encontrar en los contenidos, tanto en su versión expandida (mostrando las anclas) como en su versión reducida (sin mostrar las anclas).

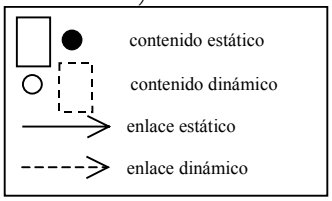

Figura 5.8 Representación gráfica de contenidos y enlaces

Si el resultado de la búsqueda hubiese sido vacío, el conjunto de anclas generadas sería el vacío, y por tanto no podría aplicarse por segunda vez la función g. En caso contrario, si volvemos a aplicar la función g sobre los contenidos anclados que acabamos de generar, obtenemos la Tabla 5.3.

k=2 g:

CA^1	$P(C \times A)$
$(resultado_1, a_1)$	$\{ (c_1, total) \}$
.....	.....
$(resultado_n, a_m)$	$\{ (c_m, total) \}$

Tabla 5.3 Segunda aplicación de la función de generación g

En este caso la función, si que es determinista, ya que cada ancla del resultado siempre enlazará con el mismo contenido. Nótese que debido a la naturaleza conjuntista de los contenidos generados, en esta última aplicación no se genera ningún contenido nuevo, ya que los contenidos c_i, son contenidos estáticos de la aplicación. En efecto si consideramos los enlaces y contenidos generados tenemos que:

$C^2 = C^1 \cup cg(CA^1) = C^1 \cup \{c_1, ..., c_n\} = C^1$
$E^2 = E^1 \cup eg(CA^1) = E^1 \cup \{ (resultado_1, a_1, c_1, total), ..., (resultado_n, a_m, c_m, total) \}$
$CA^2 = \varnothing$

La Figura 5.9 muestra los enlaces generados.

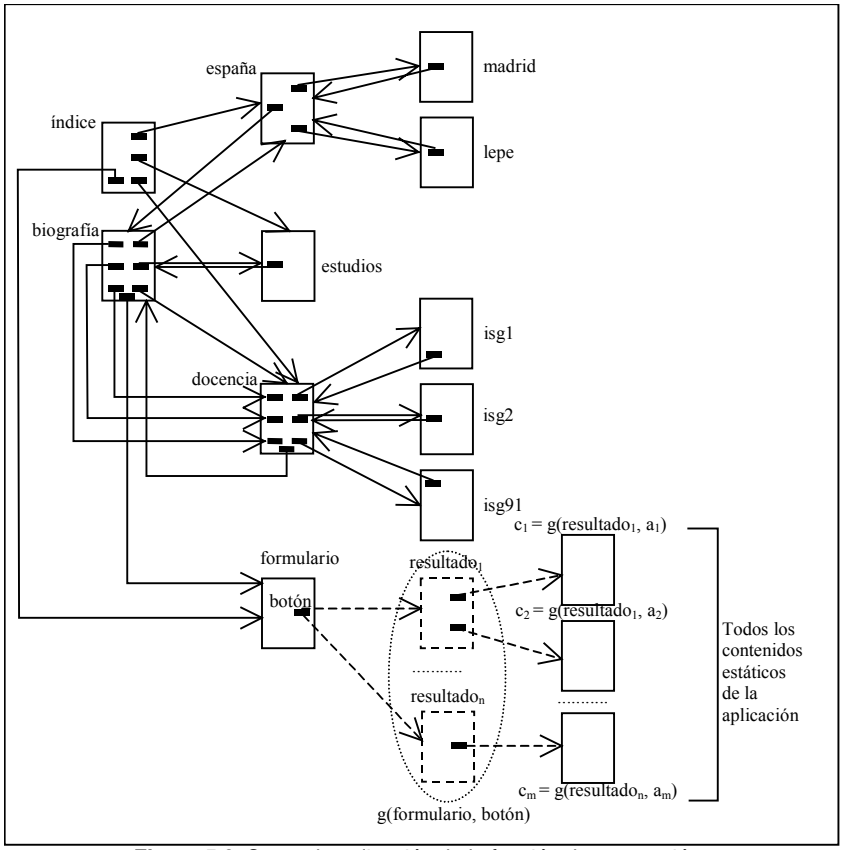

Figura 5.9 Segunda aplicación de la función de generación g

De esta forma tenemos que los contenidos de la aplicación hipermedia y los enlaces son: $C_A = C^2$, $E_A = E^2$, es decir,

C_A = { biografía, índice, españa, madrid, lepe, estudios, docencia, isg1, isg2, isg91, formulario }
$\cup$ cg(formulario, botón)

E_A = { (biografía, aEsp, españa, total), (españa, aBio, biografía, total),
(españa, aMad, Madrid, total), (Madrid, aEsp, españa, total),
(españa, aLepe, lepe, total), (lepe, aEsp, españa, total),
(biografía, aEst, estudios, total), (estudios, aBio, biografía, total),
(biografía, aDoc, docencia, total), (docencia, aBio, biografía, total),
(biografía, aDocIsg1, docencia, isg1), (biografía, aDocIsg2, docencia, isg2),
(biografía, aDocIsg91, docencia, isg91),
(docencia, aIsg1, isg1, total), (isg1, aDoc, docencia, total),
(docencia, aIsg2, isg2, total), (isg2, aDoc, docencia, total),
(docencia, aIsg91, isg91, total), (isg91, aDoc, docencia, total),
(biografía, aForm, formulario, total), (formulario, aBio, biografía, total),
(índice, aEsp, españa, total), (índice, aEst, estudios, total), (índice, aDoc, docencia, total) }
$\cup$ eg(formulario, botón) $\cup$ eg(ca(cg(formulario, botón))))

5.2.2 Esquema navegacional

Hasta ahora nos hemos limitado a modelar los contenidos de la aplicación hipermedia, bien sean estos estáticos o dinámicos. Ahora nos preocuparemos por modelar el esquema navegacional de la aplicación. Retomando la definición del Capítulo 2, este esquema describe la disposición espacial de ventanas y paneles que van a conformar dicha aplicación, así como las relaciones navegacionales existentes entre estos. Dicho de otra forma, representa la interfaz de usuario de la aplicación en su acepción hipermedia.

Para describir la información del esquema navegacional de la aplicación vamos a utilizar un grafo, al cual denominamos *extendido*. Denominamos de esta forma al grafo ya que posee diversos tipos de nodos y

Pipe:	nexo	contenedor	activador nexo
Java (Swing):	`JFrame`	`JPanel`	`JButton`
HTML:	`frameset`	`frame`	`a`
XUL:	`window`	`iframe`	`button`
PlumbingXJ:	ventana	panel	activador ventana

Tabla 5.4 Vocabulario en diversos dominios.

relaciones. Entre los tipos de nodos destacan aquellos que se encargaran de describir ventanas (nodos nexo), y los que se encargaran de describir los paneles (en la acepción de canal Amsterdam) que conforman las ventanas (nodos contenedor). Además entre estos nodos se podrán establecer distintos tipos de relaciones (arcos) para caracterizar los vínculos existentes entre los elementos que componen la interfaz de usuario. Hasta ahora hemos utilizado las expresiones ventanas, paneles y activadores de nexo. La Tabla 5.4 hace un resumen del vocabulario utilizado en distintos dominios y su relación con el aquí utilizado. Los nombres del modelo Pipe, alejados de cualquier terminología tradicional, se deben a un interés en buscar un vocabulario independiente del dominio que permita aplicarlo a diversas tecnologías. PlumbingXJ (Capítulo 6) presenta una concreción posible del vocabulario Pipe.
Como podemos ver, el esquema navegacional representa una abstracción concreta de una interfaz de usuario en la cual solo se van a considerar tres elementos. A nuestro entender estos son los de mayor importancia en el contexto hipermedia. En cualquier caso, debido a la formalización de Pipe, la ampliación del modelo seguiría un proceso bastante directo.

N: los nodos del esquema navegacional

Definimos el *conjunto de nodos del esquema navegacional*, N como:

$$N = N_c \cup N_x \cup N_a$$

donde,

- N_c: es el conjunto de *nodos contenedores*, con $N_c \cap N_x = \varnothing$, $N_c \cap N_a = \varnothing$
- N_x: es el conjunto de *nodos nexos*, con $N_x \cap N_c = \varnothing$, $N_x \cap N_a = \varnothing$
- N_a: es el conjunto de *nodos activadores de nexos*, con $N_a \cap N_x = \varnothing$, $N_a \cap N_c = \varnothing$

En base a la definición anterior, los nodos pueden ser de tres tipos distintos: contenedores, nexos y activadores de nexos. Los *nodos contenedores* son aquellos que sirven para proporcionar soporte a los contenidos de la aplicación hipermedia, es decir, son los nodos que tienen asignados contenidos. Es importante darse cuenta que los nodos contenedores pueden contener información de muy diverso tipo como, por ejemplo, texto con hiperenlaces, imágenes, un clip de audio o un proceso subordinado. Desde un punto de vista navegacional son equivalentes a paneles que aparecen dentro de ventanas.

Los nodos nexos, son los encargados de representar a las ventanas de la aplicación. Estos nodos funcionan como enumeradores de los nodos de contenidos. Es decir, los nodos nexo son los encargados de proporcionar soporte al contexto en el modelo Pipe.

Los nodos activadores de nexos sirven para activar nodos nexo desde el esquema navegacional directamente, con total independencia de los contenidos (veremos en la semántica navegacional como activar un nodo nexo

desde un contenido asignado a un nodo contenedor). Los nodos nexo se representan con un cuadrado, los nodos contenedor con un círculo, y los nodos activadores de nexos con un triángulo. La Figura 5.10 ilustra esta notación.

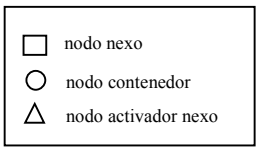

Figura 5.10 Representación visual de los nodos del modelo

A partir de ahora en adelante cuando a un nodo lo denotemos por c_i estaremos indicando que $c_i \in N_c$. Cuando a un nodo lo denotemos por x_i estaremos indicando que $x_i \in N_x$. Cuando a un nodo lo denotemos por a_i estaremos indicando que $a_i \in N_a$. Finalmente, cuando utilizamos la expresión n_i, estamos diciendo que $n_i \in (N_x \cup N_c)$.

A: el conjunto de arcos del esquema navegacional

Los arcos establecidos entre los nodos del esquema navegacional representan las relaciones existentes entre las ventanas, paneles y activadores de ventanas de la aplicación. Así definimos el *conjunto de arcos del esquema navegacional*, A:

$$A \subseteq (N \times N \times P) \cup (N \times N \times P_s \times \mathbb{R})$$

donde,

N: es el conjunto de nodos de la aplicación
P: es el tipo de arco (no sincro) entre los nodos, siendo P = { conexión, enlace }
P_s: es el tipo de arco sincro entre los nodos, siendo P_s = { conexiónS, enlaceS }
$\mathbb{R}$: es el conjunto de los números reales, indicando la información temporal para los enlaces sincro

En cuanto a la representación visual de los distintos arcos tenemos:

conexión:	línea continua
conexiónS:	línea continua decorada con la información temporal
enlace:	flecha
enlaceS:	flecha decorada con la información temporal

La Figura 5.11 recoge dicha representación gráfica.

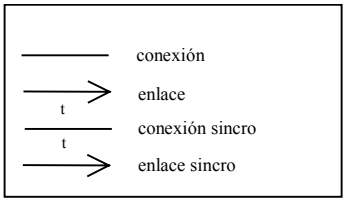

Figura 5.11 Representación visual de los arcos del

Además los elementos de A deben cumplir,

Si $a \in A$, entonces a debe ser de la forma

- $a = (x_i, x_j, \text{enlaceS}, t) \wedge \forall k((x_i, x_k, \text{enlaceS}, t) \in A \Rightarrow k = j)$
- $a = (x_i, c_j, \text{conexión}) \wedge (x_i, c_j, \text{conexiónS}, t) \notin A$
  $\wedge \neg\exists k\ ((k \neq i) \wedge ((x_k, c_j, \text{conexión}) \in A \vee (x_k, c_j, \text{conexiónS}, t) \in A))$
- $a = (x_i, c_j, \text{conexiónS}, t) \wedge (x_i, c_j, \text{conexión}) \notin A$
  $\wedge \neg\exists k\ ((k \neq i) \wedge ((x_k, c_j, \text{conexión}) \in A \vee (x_k, c_j, \text{conexiónS}, t) \in A))$
- $a = (x_i, a_j, \text{conexión}) \wedge \forall k((x_k, a_j, \text{conexión}) \in A \Rightarrow k = i)$
- $a = (c_i, c_j, \text{enlace}) \wedge \exists r(x_r, c_i, \text{conexión}) \in A \wedge \exists s(x_s, c_j, \text{conexión}) \in A$
- $a = (a_i, x_j, \text{enlace}) \wedge \forall k((a_i, x_k, \text{enlace}) \in A \Rightarrow k = j)$

Es decir las conexiones permitidas en A quedan recogidas en la Tabla 5.5 y en la Figura 5.12.

origen\destino	**nodo nexo**	**nodo contenedor**	**n. activador n.**
nodo nexo	enlace sincro *(único)*	conexión, conexión sincro *(conectado a un único nexo)*	conexión *(conectado a un único nexo)*
nodo contenedor	-	enlace	-
n. activador n.	enlace *(único)*	-	-

Tabla 5.5 Relaciones posibles entre los nodos del esquema navegacional

Es decir, si los nodos nexo representan ventanas, los contenedor paneles (virtuales o no) de esas ventanas, y los activadores de nexos botones tenemos lo siguiente. Un *enlace sincro entre dos nodos nexo* denota una relación de sincronización entre dos nodos nexo (ventana). Una *conexión entre un nodo nexo y un nodo contenedor* denota una relación de agregación, por la cual el nodo nexo (ventana) contiene al nodo contenedor (panel). Una *conexión sincro entre un nodo nexo y un nodo contenedor* es una conexión con información sobre sincronización. Una *conexión entre un nodo nexo y un nodo activador de nexo* denota una relación de agregación entre el nodo nexo (ventana) y el nodo activador de nexo (activador de ventana). Un *enlace entre un nodo contenedor y otro nodo contenedor* denota una relación navegacional entre nodos contenedor (paneles). Un *enlace entre un nodo activador de nexo y un nodo nexo* denota una relación de activación navegacional no temporal entre nodos nexo (ventanas). Nótese que tanto en la Tabla 5.5 como en las expresiones que la preceden, la relación de conexión entre un nodo nexo y nodos contenedor o activador de nexo es unidireccional. Aunque dicha relación no tiene ninguna dirección, hemos preferido mantenerla unidireccional para poder utilizar una notación más compacta. En contraposición, la relación de enlace si que es unidireccional.

También debemos indicar que todo nodo nexo debe tener conectado a un nodo contenedor, o a un activador de nexo, que todo nodo contenedor debe estar conectado a una ventana, y que todo activador de nexo debe estar conectado a una ventana y activar a otra. Las siguientes restricciones lo garantizan.

- $\forall\, x_i \exists a \in A\ (x_i = \Pi_1 a \wedge \Pi_3 a \in \{\text{ conexión, conexiónS }\})$
- $\forall\, c_i \exists a \in A\ (c_i = \Pi_2 a \wedge \Pi_3 a \in \{\text{ conexión, conexiónS }\})$
- $\forall\, a_i \exists a \in A\ \exists b \in A((a_i = \Pi_2 a \wedge \Pi_3 a \in \{\text{ conexión, conexiónS }\}) \wedge (a_i = \Pi_1 b))$

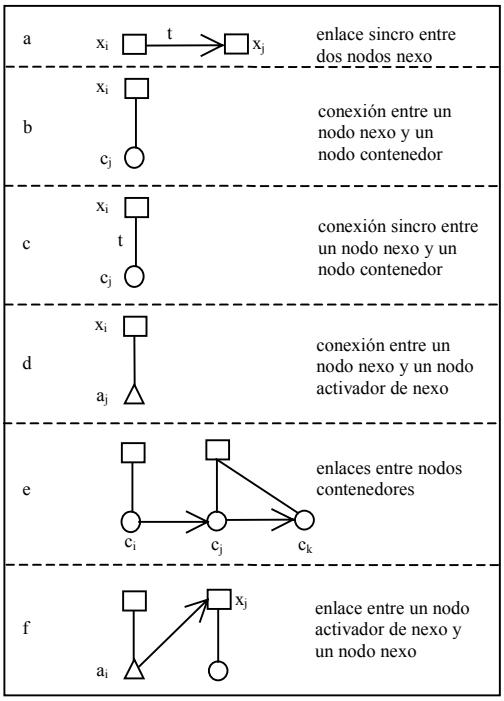

Figura 5.12 Relaciones posibles entre los nodos del modelo

Aunque aún no disponemos de conocimientos suficientes para explicar totalmente la siguiente clasificación, a lo largo de este capítulo irá tomando sentido. En efecto, en la figura anterior podemos considerar tres categorías de relaciones entre los elementos del esquema navegacional:

- *Relaciones estructurales*. Son aquellas establecidas por conexiones y conexiones sincro entre un nodo nexo y nodos contenedores y activadores de nexos. Casos b, c y d.
- *Relaciones navegacionales independientes de los contenidos*. Son aquellas que establecen un cambio de nodo nexo en función de la existencia de un enlace sincro o de la activación de un activador de nexo. Casos a y f.
- *Relaciones navegacionales que canalizan las inducidas por los contenidos*. Son aquellas que se establecen por enlaces entre nodos contenedores del mismo o de distinto nodos nexos. Caso e. A estas relaciones en particular las denominaremos *tuberías*.

Es decir, al conjunto de enlaces entre nodos contenedores lo denominaremos conjunto de *Tuberías*, T,

$$T = \{ (c_i, c_j, \text{enlace}) \in A \ \}$$

Dichas conexiones se denominan de esta forma porque ellas serán las encargadas de *canalizar* (interpretar) los enlaces entre contenidos a nivel navegacional. Es decir, si tenemos un enlace entre una biografía y una foto de España (contenidos) y queremos indicar que cuando se seleccione dicho enlace tanto la biografía como la foto deben permanecer en dos paneles distintos de una misma ventana, no tenemos más que definir una tubería entre dos nodos contenedor de un mismo nodo nexo, y canalizar el enlace de contenidos por dicha tubería. De esta forma la Figura 5.13 representa un nodo nexo (ventana) conectado con dos nodos contenedores (paneles). Además vemos como existe una tubería entre los nodos contenedor.

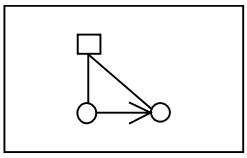

Figura 5.13 Esquema navegacional en Pipe

Con respecto a las conexiones y enlaces sincronizados, la etiqueta *t* de tiempo puede ser un valor real, o una función (aunque genérica la denominaremos h())que calcula el tiempo en función de las relaciones temporales del nodo sincro con respecto al resto de nodos contenedores del mismo nodo nexo. Es decir, supongamos:

$$a \in A,\ a = (\ x_i, c_j, \text{conexiónS}, h(c_k)) \wedge ((x_i, c_k, \text{conexión}) \in A \vee (x_i, c_k, \text{conexiónS}, t) \in A\ )$$

Entonces en función de h() tenemos:

- h() = alComienzo: $N_c \rightarrow \mathbb{R}$, alComienzo($c_k$) indica que el nodo c_j empezará a reproducirse cuando empiece a reproducirse (el contenido asignado a) el nodo c_k. Nótese que de esta forma las conexiones entre un nodo nexo y sus nodos contenedores pueden interpretarse como una extensión de la función anterior, donde los nodos contenedores empiezan a reproducirse al comienzo de su nodo nexo.
- h() = alFinal(): $N_c \rightarrow \mathbb{R}$, alFinal($c_k$) indica que el nodo c_j empezará a reproducirse cuando termine de reproducirse el nodo c_k.
- h() = respectoAlComienzo: $N_c \times \mathbb{R} \rightarrow \mathbb{R}$, respectoAlComienzo($c_k$, t) indica que el nodo c_j empezará a reproducirse t segundos después del comienzo del nodo c_k.
- h() = respectoAlFinal: $N_c \times \mathbb{R} \rightarrow \mathbb{R}$, respectoAlFinal($c_k$, t) indica que el nodo c_j empezará a reproducirse t segundos antes del final del nodo c_k.

Nótese que todas las funciones producen como resultado un real, indicando el tiempo con respecto al comienzo del nodo nexo cuando se debería empezar a reproducir los contenidos. Por otro lado si en vez de tener una conexión sincronizada tenemos un enlace sincro entre nodos nexos, las funciones son iguales, es decir, supongamos:

$$a \in A,\ a=(x_i, x_j, \text{enlaceS}, h(c_k)) \wedge ((x_i, c_k, \text{conexión}) \in A \vee (x_i, c_k, \text{conexiónS}, t) \in A$$

entonces h(), puede ser cualquiera de las funciones del caso anterior.

Con estas funciones y la semántica navegacional que describiremos a continuación, el modelo Pipe es capaz de representar relaciones de sincronización avanzadas como las descritas en [Hardman 99].

5.2.3 Funciones de canalización

Las *funciones de canalización* se encargan de modelar la relación existente entre el esquema navegacional y el grafo de contenidos, o dicho de otra forma, sirven para interpretar navegacionalmente al grafo de contenidos. Son dos funciones, la *función de asignación de contenidos* d, que asigna contenidos a los nodos contenedores, y la *función de canalización* l, que asigna enlaces de contenidos a las tuberías navegacionales.

d: la función de asignación de contenidos

La función de asignación de contenidos es la encargada de asignar contenidos a los nodos contenedor del esquema navegacional. Definimos la *función de asignación de contenidos*, d:

$$d: N_c \rightarrow (C^0 \cup \{null\}) \times \wp(C)$$

De esta forma a cada nodo contenedor se le asigna un contenido por defecto estático (o el valor null, si no hay contenido por defecto), y un conjunto de contenidos. El valor por defecto es el contenido que el nodo contenedor debe mostrar cuando sea activado. El conjunto de contenidos son los posibles contenidos que el nodo podrá mostrar a lo largo de la ejecución de la aplicación. Nótese que por definición la función d es total, ya que no tiene sentido considerar espacio para contenidos inexistentes. Además no tiene porque ser inyectiva, ni suprayectiva. No es inyectiva porque es posible que asignemos los mismos contenidos a dos nodos distintos. Respecto a no ser suprayectiva, hay que darse cuenta que razonamos en términos de $\wp(C)$ y no parece razonable esperar que existan nodos contenedores para todas las posibles combinaciones de contenidos. Esta función presenta una estrecha relación con la función de canalización de enlaces que veremos a continuación. Además la función no se define en sentido inverso debido ya que al no ser inyectiva no obtendríamos una función.

Además imponemos la condición de que el contenido por defecto de un nodo contenedor sincro no puede contener enlaces, es decir,

$$\forall c_j((x_i, c_j, \text{conexiónS}, t) \in A\ (a(\prod_1 d(c_j)) \cap \prod_2 E^0 = \varnothing)$$

La Figura 5.14 muestra de manera abstracta a la función de asignación de contenidos, sin distinguir entre el contenido por defecto y el resto de contenidos.

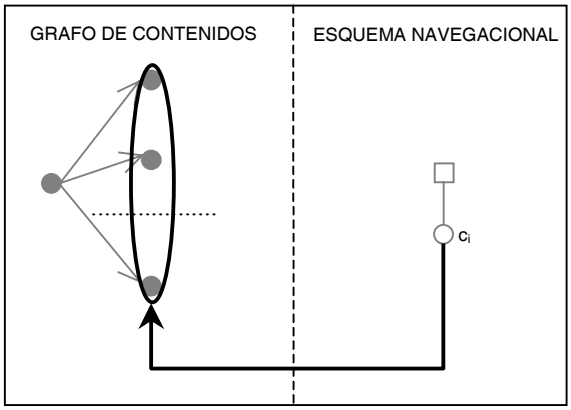

Figura 5.14 Asignación de contenidos aun nodo contenedor

l: la función de canalización de enlaces

La función de canalización de enlaces se encarga de relacionar los enlaces existentes a nivel de navegación o tuberías, T, con los enlaces existentes a nivel de contenidos, representados por E. De esta forma definimos la *función de canalización de enlaces*, l:

$$l: T \rightarrow \wp(E)$$

Además, l debe cumplir $\forall$ (c, a, c', a')$\in$ E,

- $((c, a, c', a') \in l(c_i, c_j, \text{enlace}) \wedge (c, a, c', a') \in l(c_i, c_k, \text{enlace})) \Rightarrow j = k$
- $(c, a, c', a') \in l(c_i, c_j, \text{enlace}) \Rightarrow c \in d_{1\cup2}(c_i) \wedge c' \in d_{1\cup2}(c_j)$

donde $d_{1\cup2}(c_r) = \{d_1(c_r)\} \cup d_2(c_r) = \{\prod_1 d(c_r)\} \cup \prod_2 d(c_r)$

La primera condición garantiza que el mismo enlace de contenidos no puede estar asignado a dos tuberías distintas que parten del mismo nodo. La razón es bien sencilla, ya que cuando se active el enlace en el nodo debemos ser capaces de determinar por que tubería se va a canalizar (veremos esto más adelante). La segunda condición garantiza que los contenidos origen y los contenidos destino del enlace de contenidos estén asignados como contenidos de los nodos contenedor origen y destino de la tubería que canaliza dicho enlace. La Figura 5.15 muestra de manera abstracta a la función de canalización de enlaces.

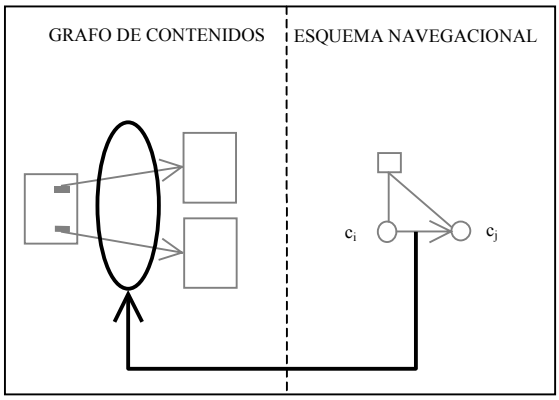

Figura 5.15 Canalización de enlaces

La función de canalización determina por que enlace navegacional van a pasar los diversos enlaces a nivel de contenidos, por eso razonamos con $\wp(E)$, y no con E, ya que es posible (y de hecho normal) que la misma tubería canalice más de un enlace. Esta función es una de las partes más importantes del modelo Pipe y por eso le prestaremos una especial atención. El modelo parte de una escrupulosa separación entre el nivel de contenidos y sus enlaces, y entre el nivel de navegación y sus enlaces. Con esto se obtiene el beneficio de poder razonar con dos niveles ortogonales: contenidos y navegación de los mismos. El inconveniente de esta aproximación es tener que ligar contenidos a su navegación, y en concreto a como vamos a interpretar los enlaces que poseen los contenidos. Por ejemplo, pensemos en el ejemplo del sitio Web del profesor (descrito en su totalidad en la Figura 5.9). En este caso nos restringiremos a la biografía, la información sobre los estudios y la docencia. Aunque existen enlaces entre la biografía y los estudios y docencia, la existencia de dichos enlaces no determina como van a ser presentados en la interfaz de usuario. Una posibilidad es que la biografía aparezca en el lado izquierdo de la ventana, y cuando seleccionamos la facultad o carrera, la descripción sobre esta aparece en el lado derecho de la misma ventana. Otra opción es que en una ventana aparezca la biografía y cuando seleccionamos la facultad o carrera, la descripción sobre esta aparezca en otra ventana. Veamos como representa el modelo Pipe estas dos opciones, obviando los enlaces entre los estudios y la docencia con la biografía.

En ambos casos los conjuntos C_A, y E_A no varían. En efecto:

C_A = { biografía, estudios, docencia }
E_A = { (biografía, aEstudios, estudios, total), (biografía, aDoc, docencia, total) }

Estos conjunto indican tres contenidos, y dos enlaces. Si ahora queremos representar una navegación con una sola ventana tenemos:

N = { x1, c1.1, c1.2 }
A = { (x1, c1.1, conexión), (x1, c1.2, conexión), (c1.1, c1.2, enlace) }

es decir, el grafo de la Figura 5.16.

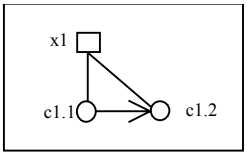

Figura 5.16 Representación gráfica del esquema navegacional

Ahora que tenemos definido el esquema navegacional, podemos asignar contenidos a los nodos. En este caso la biografía va en el nodo c1.1, y la información sobre carrera y universidad está asignada al nodo c1.2. En efecto (a partir de ahora vamos a utilizar la notación (**x**, h(x)) para representar a una función h),

d = { (**c1.1**, biografía, ∅), (**c1.2**, null, { estudios, docencia }) }

Hemos asignado a c1.1 como contenido por defecto biografía, y a c1.2 el valor null porque queremos que cuando se acceda al nodo nexo x1, se muestre la biografía exclusivamente. Este hecho enlaza con la semántica de presentación que definiremos formalmente en el siguiente apartado.

Por último solo queda canalizar los enlaces del nivel de contenidos por tuberías. En efecto,

l = { (**c1.1, c1.2, enlace** , { (biografía, aEst, estudios, total), (biografía, aDoc, docencia, total) }) }

Es decir, la función de canalización de enlaces lo único que indica es que los enlaces a nivel de contenidos se corresponden con el arco/enlace que conecta los nodos c1.1 y c1.2. Podemos decir que hemos "canalizado" los enlaces a nivel de contenidos por la "tubería" que conecta c1 y c2. De aquí el nombre de Pipe para el modelo, y el nombre de la función l. La Figura 5.17 ilustra esta relación.

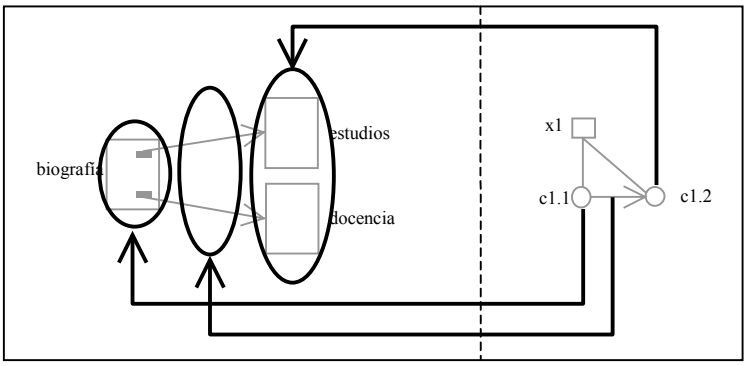

Figura 5.17 Asignación de contenidos y enlaces al esquema navegacional

Si ahora queremos indicar que tenemos dos ventanas en vez de una, no hace falta cambiar ni C_A, ni E_A. En efecto, ahora N y A son:

N = { x1, c1.1, x2, c2.1 }
A = { (x1, c1.1, conexión), (x2, c2.1, conexión), (c1.1, c2.1, enlace) }

Ahora tenemos un arco enlace entre los nodos c1.1 y c2.1, indicando que los contenidos de c2.1 se muestran como resultado de respuesta de un enlace de la biografía con el resto de información. Gráficamente tenemos la Figura 5.18.

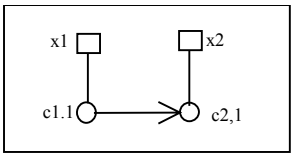

Figura 5.18 Esquema navegacional alternativo

La función d es:

$$d = \{ (\mathbf{c1.1}, \text{biografía}, \varnothing), (\mathbf{c2.1}, \text{null}, \{\text{estudios, docencia}\}) \}$$

La función l tiene la forma:

$$l = \{ (\mathbf{c1.1, c2.1, enlace}, \{(\text{biografía, aEst, estudios, total}), (\text{biografía, aDoc, docencia, total})\}) \}$$

La Figura 5.19 ilustra las nuevas relaciones.

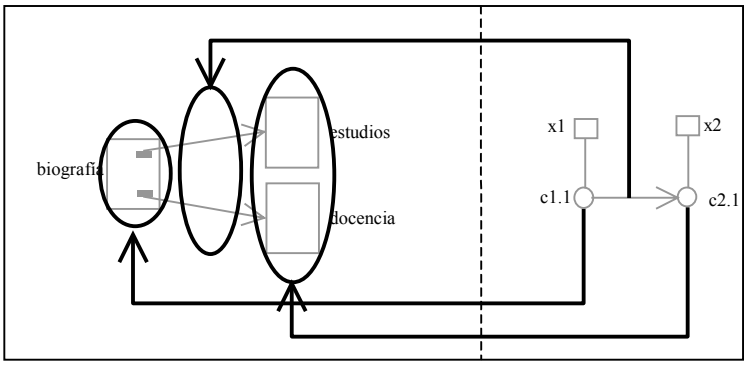

Figura 5.19 Asignación de contenidos y enlaces al esquema navegacional

En el supuesto de tener una aplicación dinámica, las funciones d y l se comportan de la misma manera. En efecto, volviendo el ejemplo del formulario de búsqueda en el sitio Web. En esta ocasión queremos que tras iniciar la búsqueda, la hoja de resultados se muestre en un panel de una ventana distinta a la que contiene al formulario de búsqueda. En otro panel de esa misma ventana se mostrarán los contenidos accedidos desde el resultado de la búsqueda. En este caso tenemos:

N = { x1, c1.1, x2, c2.1, c2.2 }
A = { (x1, c1.1., conexión), (x2, c2.1, conexión), (x2, c2.2., conexión),
(c1.1, c2.1, enlace), (c2.1, c2.2, enlace), (c2.2, c2.2, enlace) }

La Figura 5.20 recoge este esquema.

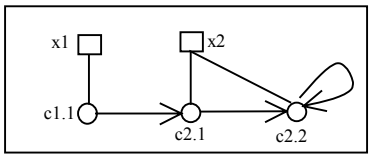

Figura 5.20 Esquema navegacional

Ahora las funciones d y l deberán establecerse en base a la función g. Tenemos,

d = { (**c1.1**, formulario, ∅),
(**c2.1**, null, cg(formulario, botón)), (**c2.2**, null, cg(ca(resultado)) ∪ C) }

supuesto que resultado = cg(formulario, botón).

Es decir, el nodo c1.1 contiene el formulario de búsqueda. El nodo c2.1 contiene el resultado de la misma, y el nodo c2.2 contiene los contenidos con los que enlaza el resultado. Nótese que en nodo c2.2 también

contienen todos los contenidos. Esto se debe a que es posible que en el nodo aparezcan contenidos con enlaces que pueden llevarnos a cualquier contenido estático. Para indicar que esto es posible se asigna el conjunto de contenidos. Nótese que no es necesario que aparezcan todos, pero ahora la validación de contenidos asignados al nodo siempre es positiva. Podemos permitirnos esta falta de precisión porque precisamente ahora queremos indicar que solo tenemos un panel activo, y que este panel va a mostrar toda la información.

Al igual que con la función de asignación de contenidos debemos tener en cuenta los enlaces susceptibles de ser activados por todos los contenidos que se muestran en el panel. Por esto ahora necesitamos canalizar todos los enlaces por una tubería. Esto indica que no hace falta validar la canalización (o que siempre es cierta) ya que tenemos una única tubería. De esta forma definimos la función l,

l = { (**c1.1, c2.1, enlace**, eg(formulario, botón)),
(**c2.1, c2.2, enlace**, eg(ca(resultado))),
(**c2.2, c2.2, enlace**, E)

Es decir, la primera tubería canaliza en enlace generado entre el formulario y el resultado. La segunda canaliza los enlaces generados entre el resultado y los contenidos a los que referencia. finalmente la última tubería canaliza cualquier enlace activado desde los contenidos accesibles desde **resultado**. La Figura 5.21 ilustra estos conceptos.

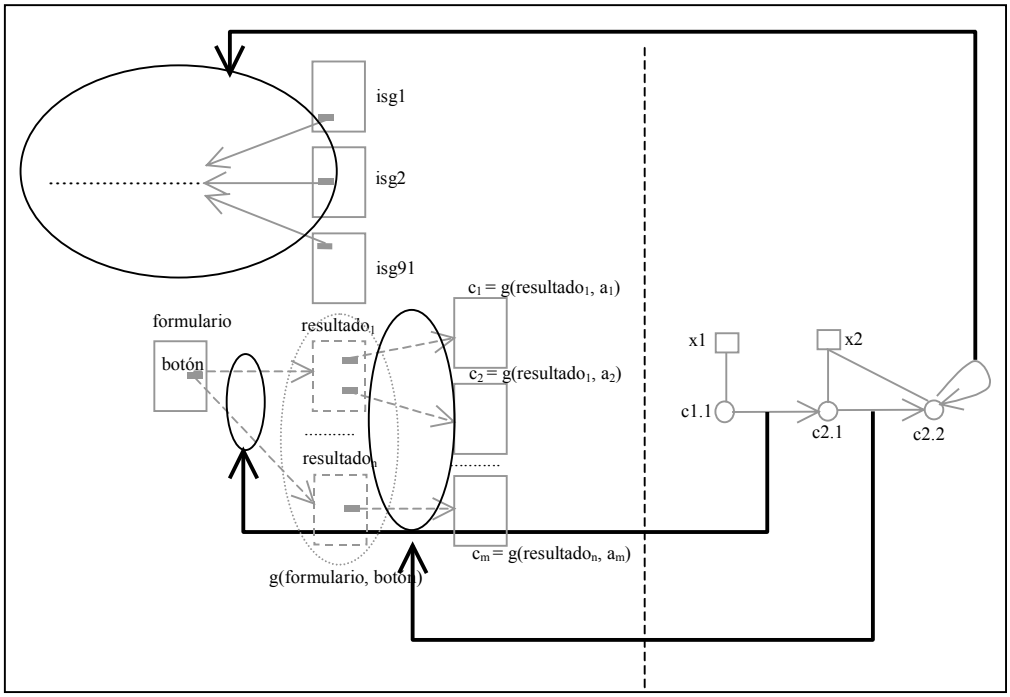

Figura 5.21 Asignación de enlaces al esquema navegacional. Hemos obviado la asignación de contenidos porque esta se puede deducir de la canalización de enlaces. Nótese que por definición de enlazados el nodo c2.2 tiene asignados todos los contenidos de la aplicación. De la misma forma, por definición de enlaces, la tubería (c2.2, c2.2, enlace) tiene asignadas todos los enlaces de la aplicación.

La función l es total, pero no es inyectiva ni suprayectiva. Es total porque no tiene sentido considerar tuberías que no van a canalizar ningún enlace de contenidos. No es inyectiva ya que aunque cada tubería solo puede canalizar un conjunto de enlaces de contenido, estos pueden estar asignados a tuberías diferentes (lo que representa distintas interfaces para los mismos contenidos). Veremos este supuesto más adelante en un ejemplo. No es suprayectiva al trabajar con $\wp$(E). Además la función no se define en sentido inverso ya que al no ser inyectiva no sería función.

Aunque en este capítulo vamos a ver un ejemplo de las posibilidades expresivas del modelo, en el Capítulo 7 también se pueden encontrar varios ejemplos de aplicación del mismo. Otro ejemplo puede ser un documento Word con enlaces. En este caso tenemos un único contenido autoenlazado, quedando canalizados todos los enlaces por una única tubería. El esquema navegacional sería similar al descrito en la

Figura 5.43 de este capítulo, donde la ventana de presentación estaría representada por un nodo nexo conectado a un único nodo contenedor. La posibilidad de incluir *scroll* en el panel que contiene al documento Word se obvia en el modelo, al movernos a otro nivel de abstracción. Por supuesto aplicaciones concretas del mismo (como la presentada en el Capítulo 6) deben proporcionar alguna solución (en este caso todos los paneles tienen *scroll*). El modelo también obvia otras caracterizaciones, como la de los índices desplegables, quedando caracterizados estos por un único contenido con todas las anclas que contenga el menú.

p: la función de presentación para los nodos y contenidos

La estructura de nodos nexo y nodos contenedores son una abstracción de ventanas formadas por paneles, dentro de las cuales aparecen contenidos y enlaces. Este es el denominado esquema navegacional, ya que indica que componentes aparecen, en que contexto, y con que enlaces. Dicho esquema no determina la posición concreta de los paneles dentro de las ventanas, ni la fuente, color o estilo del texto, etc. Pues bien las especificaciones de presentación se encargan de determinar estos factores. Es decir, la función de presentación es una abstracción de los parámetros de presentación de una interfaz gráfica actual (al igual que la semántica de navegación es una abstracción de la interfaz de usuario). Para asignar especificaciones de presentación a cada nodo utilizamos la función p, en efecto:

$$p: N \rightarrow EP$$

donde,

- N: es el conjunto de nodos
- EP: es el conjunto de especificaciones de presentación para cada nodo.

Nótese que los nodos funcionan como contenedores abstractos de información, indicando únicamente que contenidos aparecen dentro de que ventanas, y que enlaces parten de estos contenidos. Es decir solo determinan la estructura navegacional de la aplicación. Para fijar la estructura presentacional completa necesitamos la función p y las especificaciones de presentación.

Tal y como hemos definido la función p, esta se encarga de determinar la posición espacial de los nodos del modelo, y una presentación por defecto para los contenidos. También podemos considerar una presentación especial para los contenidos que tuviese en cuenta la organización estructural de estos. En este caso también llamamos a esta función p. En efecto,

$$p: C \rightarrow EPC$$

En este caso a cada contenido se le aplica una especificación de presentación de contenido, cuya complejidad puede variar según la naturaleza de los mismos. Por ejemplo, en casos concretos se podrían proporcionar especificaciones de presentación similares a hojas de estilo CSS o similares. También es importante darse cuenta que los nodos contenedor pueden tener asignados procesos subordinados. En este caso podrían proporcionarse construcciones similares a las propuestas en [Horrocks 99].

Hemos obviado una caracterización en mayor detalle de esta función ya que su utilidad es parcial en labores de conceptualización. De todas formas se incluye para mostrar su viabilidad de uso en diseño. Nótese por tanto, que con las funciones de presentación no se intenta capturar todas las posibilidades de una interfaz gráfica de usuario actual. Simplemente se pretende dar un soporte básico a las mismas. La razón se debe al uso de Pipe como herramienta de conceptualización y prototipado. En la medida que se desease utilizar Pipe como herramienta de diseño y construcción (induciendo por tanto una metodología) sería deseable ampliar la potencia expresiva de estas funciones de presentación.

5.2.4 Semántica de navegación y semántica de presentación

Por ahora tenemos mecanismos para describir la estructura de enlaces entre contenidos, el esquema navegacional de la aplicación, y las relaciones existentes entre ambos. Solo nos falta describir el comportamiento dinámico de las aplicaciones, comportamiento que hemos ido introduciendo de manera informal en los ejemplos anteriores. Lo que queremos formalizar es el siguiente comportamiento navegacional extraído de las relaciones existentes entre los nodos del esquema navegacional (Tabla 5.5). Es decir, si los nodos nexo representan ventanas, los contenedor paneles (virtuales o no) de esas ventanas, y los

activadores de nexos representan botones tenemos lo siguiente Un *enlace sincro entre dos nodos nexo* denota una relación de sincronización entre dos nodos nexo (ventana). De esta forma cierta cantidad de tiempo después de acceder al nodo origen se debe activar el nodo nexo (ventana) destino. Una *conexión entre un nodo nexo y un nodo contenedor* denota una relación de agregación, por la cual el nodo nexo (ventana) contiene al nodo contenedor (panel). De esta forma cuando se acceda al nodo nexo, los nodos contenedores mostrarán sus contenidos por defecto. Una *conexión sincro entre un nodo nexo y un nodo contenedor* es una conexión con información sobre sincronización. De esta forma cuando se acceda al nodo nexo, los nodos contenedores mostrarán sus contenidos por defecto de acuerdo a la información de sincronización. Una *conexión entre un nodo nexo y un nodo activador de nexo* denota una relación de agregación entre el nodo nexo (ventana) y el nodo activador de nexo (activador de nexo). De esta forma el nodo activador de nexo se activará cuando sea accedido el nodo nexo. Un *enlace entre un nodo contenedor y otro nodo contenedor* denota una relación navegacional entre nodos contenedor (paneles). De esta forma cuando se active un enlace de un contenido asignado al nodo contenedor origen, el destino de dicho enlace se deberá mostrar en el nodo contenedor destino. Un *enlace entre un nodo activador de nexo y un nodo nexo* denota una relación de activación no temporal entre nodos nexo (ventanas). De esta forma cuando el nodo activador de nexo sea seleccionado se activará el nodo nexo destino.

Formalicemos pues dicha información. La ventaja derivada de dicha formalización será la de disponer de semántica de presentación programable e independiente del código [Stotts 89], [Heath99], lo cual permitirá la generación automática de prototipos/aplicaciones como sucedía con la semántica αTrellis proporcionada por el modelo Trellis. Aunque totalmente formalizada, esta semántica de navegación está más en la línea del trabajo presentado por [Millard 00] que por [Stoy 77]. Queda como trabajo futuro de esta tesis un estudio en profundidad de [Stoy 77], y de como pueden afectar las ideas ahí expuestas al modelo Pipe.

SN: la semántica navegacional del modelo

Utilizando la formalización estática del modelo vamos a definir la semántica de navegación del modelo, es decir, que cambios se suceden en la interfaz cuando se activa un enlace. Para esto introduciremos los conceptos de nodo activo, activación de un nodo nexo, y activación de un enlace. Es decir la *semántica navegacional*, SN, es una tupla:

$$SN = (a, f)$$

donde,

- a: es la *función de activación de un nodo nexo*. Esta función se encarga de caracterizar los cambios producidos en el esquema navegacional a raíz de la activación de un nodo nexo.
- f: es la *función de activación de un enlace*. Esta función se encarga de caracterizar los cambios producidos en el esquema navegacional a raíz de la selección de un enlace de contenidos, la activación por tiempo de un nodo nexo, o la selección de un nodo activador de ventana.

Antes de definir las funciones anteriores introduciremos el concepto de *nodo activo*. Un nodo activo no es más que un nodo que está mostrando sus contenidos (si es un nodo contenedor), o un nodo que se hace visible al usuario (activador de nexo o ventana). Los caracterizaremos como un simple conjunto, denominado Activos, y los representaremos por un punto dentro del nodo, o marcando el nombre del nodo activo en negrita (en este texto optaremos por la segunda alternativa). No solo nos interesa considerar que nodos del esquema navegacional están activos, sino además que contenidos están reproduciendo dichos nodos activos que sean contenedores. A estos nodos los representaremos a través del conjunto Mostrar. Definimos ambos conjuntos como:

- $\text{Activos} \subseteq V = N \cup (N \times \mathbb{R})$. Indica en cada momento cual es el conjunto de nodos activos, así como información temporal.
- $\text{Mostrar} \subseteq M = (N_c \times C) \cup (N_c \times C \times \mathbb{R})$. Indica que contenidos exhibe cada nodo contenedor del nodo nexo activo, así como información sobre sincronización si fuera necesario.

Lo primero de todo debemos indicar que el inicio de la navegación se va a producir por un nodo nexo determinado (es decir, por una ventana determinada). Este hecho se corresponde con activar el nodo nexo inicial. Para activar nodos nexo disponemos de la función de activación a,

$$a: N_x \rightarrow \wp(V) \times \wp(M)$$

de tal forma que $a(x_i) = (\text{Activos}_n, \text{Mostrar}_n)$

donde,

$$\begin{aligned} Activos_n = \{x_i\} &\cup \{c_j \mid (x_i, c_j, \text{conexión}) \in A\} \\ &\cup \{a_j \mid (x_i, a_j, \text{conexión}) \in A\} \\ &\cup \{c_j \mid (x_i, c_j, \text{conexiónS}, t) \in A\} \\ &\cup \{(x_j, t) \mid (x_i, x_j, \text{enlaceS}, t) \in A\} \end{aligned}$$

$$\begin{aligned} Mostrar_n = &\{(c_j, d_l(c_j)) \mid (x_i, c_j, \text{conexión}) \in A\} \\ &\cup \{(c_j, d_l(c_j), t) \mid (x_i, c_j, \text{conexiónS}, t) \in A\} \end{aligned}$$

Es decir, cuando se activa un nodo nexo se activan todos los nodos contenedores y los nodos activadores de nexos. También se activan los nodos contenedores con conexión sincro (paneles de la misma ventana), y los nodos nexo con enlace sincro (otra ventana distinta) quedan pendientes de activación, según su información temporal. Además todos los nodos contenedores muestran su contenido por defecto, teniendo en cuenta, si fuera necesario, restricciones temporales. Nótese que para que esta semántica funcione, es necesario que el componente encargado de reproducir los contenidos, debe ser capaz de llevar cuenta de la información temporal. Hemos obviado este componente, ya que lo único que hace es reproducir un contenido si no tiene información temporal, y esperar a que se cumpla la restricción correspondiente para reproducirlo. Es decir, realmente faltaría una función reproducir,

$$\begin{aligned} &\text{reproducir}: (C \cup (C \times \mathbb{R} \times N_x)) \rightarrow \Omega \\ &\text{reproducir}(c) = \omega(c) \\ &\text{reproducir}(c, t, x_i) = \omega(c), \text{ si } t = \text{tiempo}(x_i) \end{aligned}$$

Donde lo único que hace esta función es reproducir el contenido ($\omega(c)$) si no hay restricciones temporales, y reproducirlo acorde a estas restricciones temporales si las hay ($\omega(c)$, si $t = \text{tiempo}(x_i)$, supuesto que la función $\text{tiempo}(x_i)$ informe de la cantidad de unidades de tiempo transcurridas desde la activación del nodo nexo x_i).Como toda la información que necesita esta función se encuentra en el conjunto $Mostrar_n$, obviamos su inclusión. Además dicha función debería tener acceso (incluirse en su dominio) a los nodos contenedores, y a la función de asignación de contenidos para garantizar que puede llevar cuenta de las restricciones temporales impuestas.

Ahora debemos indicar que sucede cuando se activa un enlace de contenidos. Con este fin definimos la función de activación de enlaces f,

$$f: ((C_{ns} \times A) \cup \{\perp\}) \times N \times \wp(V) \times \wp(M) \rightarrow \wp(V) \times \wp(M)$$

Donde el símbolo $\perp$ se utiliza para los enlaces que no tienen origen en enlaces de contenido (es decir los sincro entre nodos nexo, y los de activador de nexo). A la hora de definir la función f hay dos tipos de situaciones que debemos tener en cuenta.

- La función f debe actualizar el esquema navegacional a raíz de la selección de un *enlace de contenidos*. Esta situación queda recogida en las figuras 5.22 y 5.24. En este caso dicha función f, tendrá que recurrir a la función de asignación de contenidos d, a la función de canalización l, y a los enlaces de contenidos definidos en base a la función de relación r para caracterizar los cambios.
- La función f debe actualizar el esquema navegacional a raíz de la *activación por tiempo de un nodo nexo* o la *selección de un activador de nexo*. Dicho de otra forma, debe responder a un cambio en el esquema navegacional que no corresponde con la selección de un enlace de contenidos. De ahí la necesidad de incluir $\perp$. Esta situación queda recogida por las figuras 5.26 y 5.28. En este caso dicha función f solo tendrá que recurrir a la función de asignación de contenidos para caracterizar los cambos.

En base a lo anterior definimos a la función f de la siguiente manera.

- El origen se produce en un nodo de contenidos, y el destino es otro nodo de contenidos, es decir,

  $$f((o, a), c_i, Activos_n, Mostrar_n) = (Activos_{n+1}, Mostrar_{n+1})$$

  $$y \ \exists j \ ((c_i, c_j, \text{enlace}) \in A \wedge \text{enlAct}(o, a) \in l(c_i, c_j, \text{enlace})) \wedge (c_i, o) \in Mostrar_n$$

Nótese que para garantizar que la primera parte de esta condición anterior se pueda validar, es necesario que el mismo enlace de contenidos no se asigne a dos tuberías distintas que partan del mismo nodo de contenidos c_i (cosa que tenemos garantizada por definición de la función l). Por otro lado está condición lo único que exige es que exista una tubería con origen en el nodo en el que se activa el enlace, y con destino en otro nodo de contenidos que el enlace de contenidos que estamos activando esté canalizado por esa tubería. La segunda parte de esta condición exige que realmente estemos mostrando el contenido del cual seleccionamos el enlace (nótese que esto garantiza que el nodo del cual parte el enlace esté activo). No es posible que se produzca la selección de un enlace en un nodo contenedor con conexión sincro porque los contenidos iniciales de estos nodos no pueden tener enlaces, y tampoco pueden llegar tuberías a estos nodos. Veamos que sucede si el nodo destino pertenece o no al mismo nodo nexo, es decir,

- Si $\exists r \exists s((x_r, c_i, \text{conexión}) \in A \wedge (x_s, c_j, \text{conexión}) \in A \wedge r \neq s)$, es decir, tenemos una situación como la descrita en la Figura 5.22.

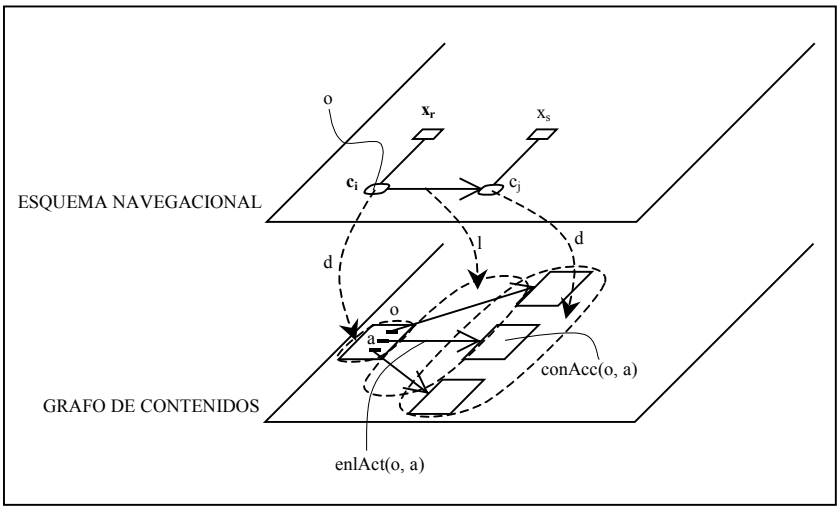

Figura 5.22 Esquema navegacional, grafo de contenidos y relaciones entre ambos antes de aplicar la función f

$$Activos_{n+1} = \prod_1 a(x_s)$$

$$\begin{aligned} Mostrar_{n+1} = \; & \{ (c_j, \text{conAcc}(o, a)) \} \\ & \cup \{ (c_k, d_1(c_k)) \mid (x_s, c_k, \text{conexión}) \in A \wedge k \neq j \} \\ & \cup \{ (c_k, d_1(c_k), t) \mid (x_s, c_k, \text{conexiónS}, t) \in A \wedge k \neq j \} \end{aligned}$$

Es decir, se activa el nodo nexo que contiene al nodo contenedor destino de la tubería, y todos los nodos muestran sus contenidos por defecto, salvo el nodo contenedor destino que muestra el destino del enlace navegacional a través de la función conAcc. En el caso enlaces estáticos devuelve el extremo del enlace. En caso de enlaces dinámicos devuelve el contenido generado. Además, esta definición es consistente gracias a la definición de la función l, la cual a su vez tiene en cuenta a la función d. Nótese que al desestimar $Activos_n$ y $Mostrar_n$ "eliminamos" la ventana de la que partió el enlace. Además, aunque en $Mostrar_n$ obviamos la información sobre el ancla destino, esta se incluye en $\prod_2 r(o, a)$. Por último, los nodos con conexión sincro tienen información temporal sobre cuando deben ser activados. La Figura 5.23 ilustra los cambios en el esquema navegacional.

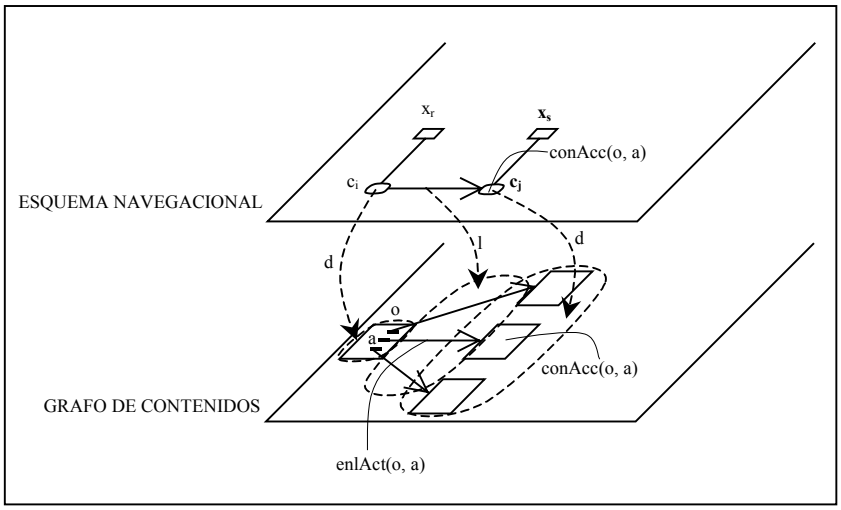

Figura 5.23 Esquema navegacional, grafo de contenidos y relaciones entre ambos después de aplicar la función f

- Si $\exists r((x_r, c_i, \text{conexión}) \in A \wedge (x_r, c_j, \text{conexión}) \in A)$, es decir, tenemos una situación como la descrita en la Figura 5.24.

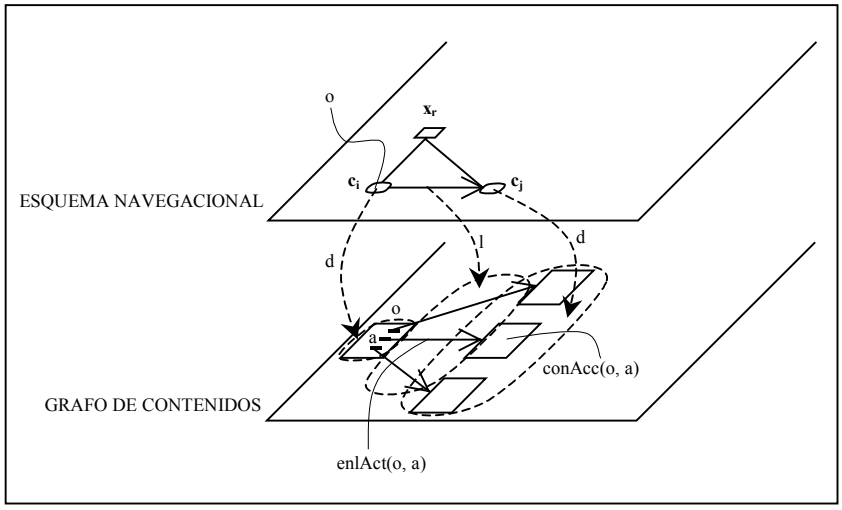

Figura 5.24 Esquema navegacional, grafo de contenidos y relaciones entre ambos antes de aplicar la función f

$$Activos_{n+1} = Activos_n$$

$$Mostrar_{n+1} = Mostrar_n \setminus \{ (c_j, m) \mid (c_j, m) \in Mostrar_n \} \cup \{ (c_j, conAcc(o, a)) \}$$

Es decir, se activa el nodo contenedor correspondiente y se muestra el contenido seleccionado por el enlace (bien estático o dinámico) dejando de mostrar el contenido previo. La Figura 5.25 ilustra los cambios en el esquema navegacional.

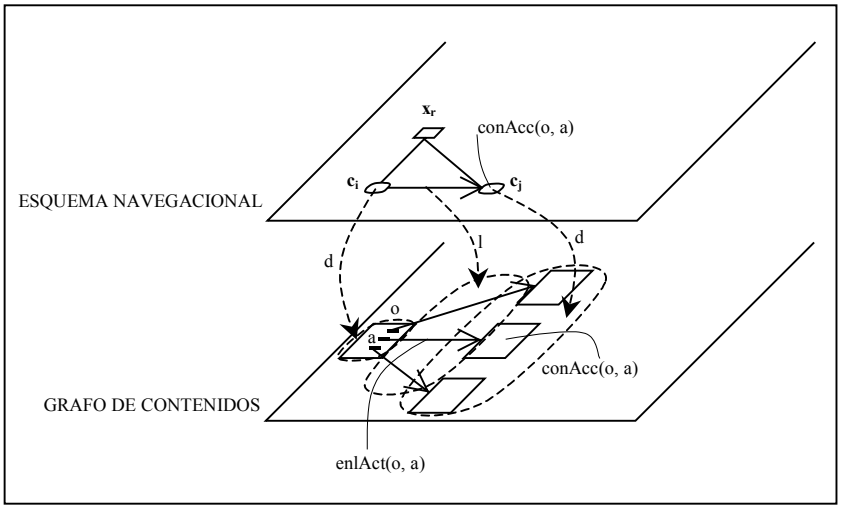

Figura 5.25 Esquema navegacional, grafo de contenidos y relaciones entre ambos después de aplicar la función f

- El origen se produce en un nodo nexo, y por tanto es un enlace sincro. Nótese que en este caso no tenemos enlace de contenidos y usamos ⊥. Es decir, estamos en una situación descrita como en la Figura 5.26.

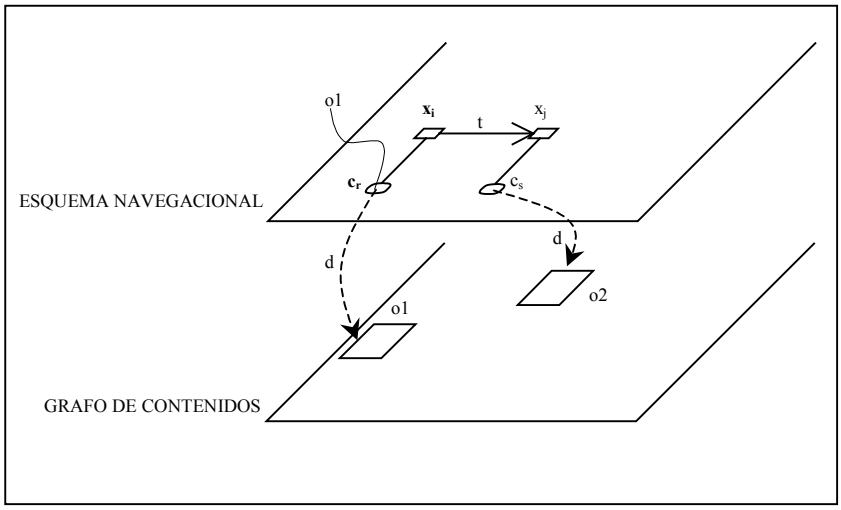

Figura 5.26 Esquema navegacional, grafo de contenidos y relaciones entre ambos antes de aplicar la función f

$f(\perp, x_i, Activos_n, Mostrar_n) = a(x_j)$
si $((x_j, t) \in Activos_n \wedge t = tiempo(x_i))$

Donde la función tiempo(x_i) devuelve el tiempo transcurrido desde la activación del nodo nexo x_i. Nótese que es necesario que solo parta un enlace sincro del nodo origen, y que nos limitamos a activar el nodo destino cuando han transcurrido t segundos desde la activación de x_i. Nótese que si se produce simultáneamente la selección de un enlace y la activación de un nexo sincro es necesario que el sistema discretice los eventos y responda a ellos consistentemente. La Figura 5.27 ilustra los cambios en el esquema navegacional.

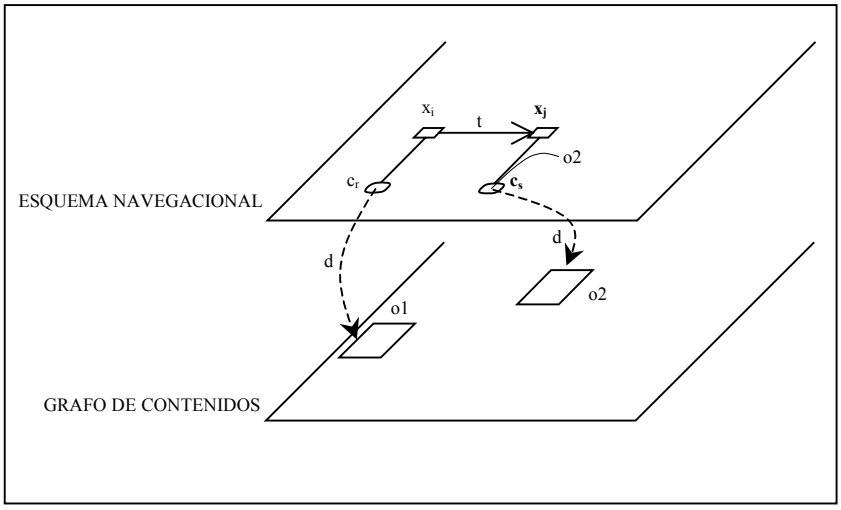

Figura 5.27 Esquema navegacional, grafo de contenidos y relaciones entre ambos después de aplicar la función f

- El origen se produce en un activador de nexo. Nótese que en este caso no tenemos enlace de contenidos y usamos ⊥. Es decir, estamos en una situación como la descrita en la Figura 5.28.

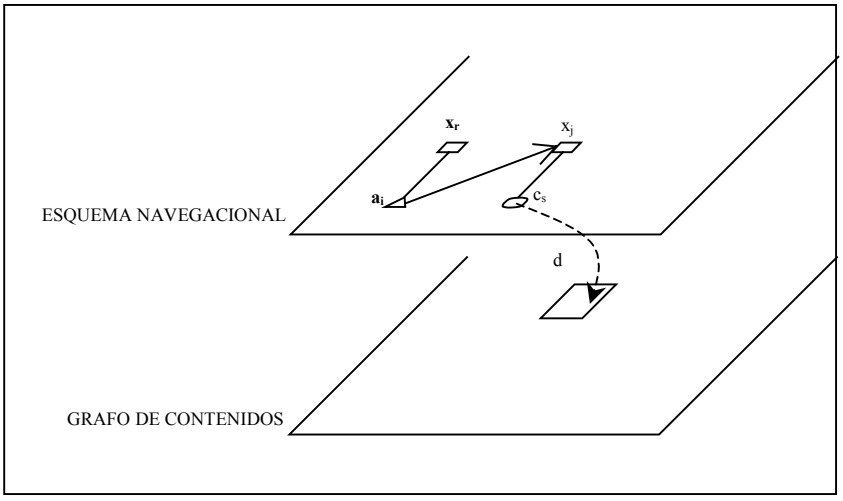

Figura 5.28 Esquema navegacional, grafo de contenidos y relaciones entre ambos antes de aplicar la función f

$f(\perp, a_i, Activos_n, Mostrar_n) = a(x_j)$ si $\exists$ j $((a_i, x_j, enlace) \in A) \wedge a_i \in Mostrar_n$

En este caso simplemente activamos el nodo destino. Nótese que por como hemos definido A, los activadores de nexos solo pueden enlazar con un destino. La Figura 5.29 ilustra los cambios en el esquema navegacional.

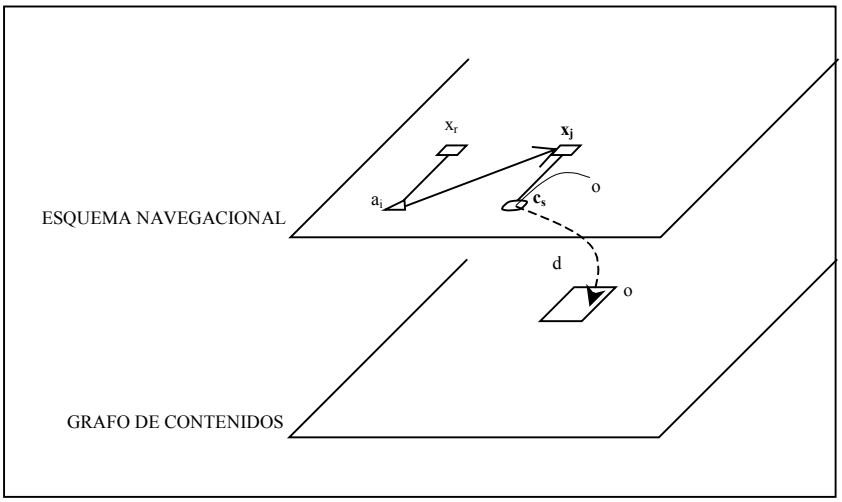

Figura 5.29 Esquema navegacional, grafo de contenidos y relaciones entre ambos después de aplicar la función f

- En otro caso, f se comporta como la identidad. Nótese que este último punto convierte en total a la función f.

Nótese que la semántica de navegación no considera la opción de "ventana anterior". Esta es una opción del navegador que está visualizando los contenidos, pero no se trata de un enlace de contenidos, ni navegacional (volver a una ventana cuyos contenidos se originaron por la activación de un hiperenlace supone conocer cual fue en enlace que originó la presentación de esos contenidos). De todas formas hay una elegante solución para representar la "marcha atrás" en el modelo Pipe: seleccionar el estado del modelo anterior a la visualización de la ventana, es decir, el conjunto de $Activos_n$, $Mostrar_n$, C^n, y E^n (nótese que es posible que hayamos generado dinámicamente enlaces y contenidos), que estaban en la transición n-ésima que nos llevó a la ventana a la cual queremos retroceder.

Ejemplo

Retomemos ahora el ejemplo que nos servía de hilo conductor en el capítulo de sistemas de representación hipermedia. Como ya vimos en el apartado sobre contenidos y relaciones tenemos que:

C_A = { biografía, índice, españa, madrid, lepe, estudios, docencia, isg1, isg2, isg91, formulario, resultado }
∪ {imagen, audio}

E_A = { (biografía, aEsp, españa, total), (españa, aBio, biografía, total),
(españa, aMad, Madrid, total), (madrid, aEsp, españa, total), (españa, aLepe, lepe, total),
(lepe, aEsp, españa, total),
(biografía, aEst, estudios, total), (estudios, aBio, biografía, total),
(biografía, aDoc, docencia, total), (docencia, aBio, biografía, total),
(biografía, aDocIsg1, docencia, isg1), (biografía, aDocIsg2, docencia, isg2),
(biografía, aDocIsg91, docencia, isg91),
(docencia, aIsg1, isg1, total), (isg1, aDoc, docencia, total),
(docencia, aIsg2, isg2, total), (isg2, aDoc, docencia, total),
(docencia, aIsg91, isg91, total), (isg91, aDoc, docencia, total),
(biografía, aForm, formulario, total), (formulario, aBio, biografía, total),
(índice, aEsp, españa, total), (índice, aEst, estudios, total), (índice, aDoc, docencia, total) }
∪ eg(formulario, botón)) ∪ eg(ca(resultado)))

Es decir nos encontramos con la situación descrita en la Figura 5.30, similar a la Figura 5.9, y supuesto que concretamos los eg(ca(resultado)). En esta figura hemos optado por eliminar la información sobre las anclas.

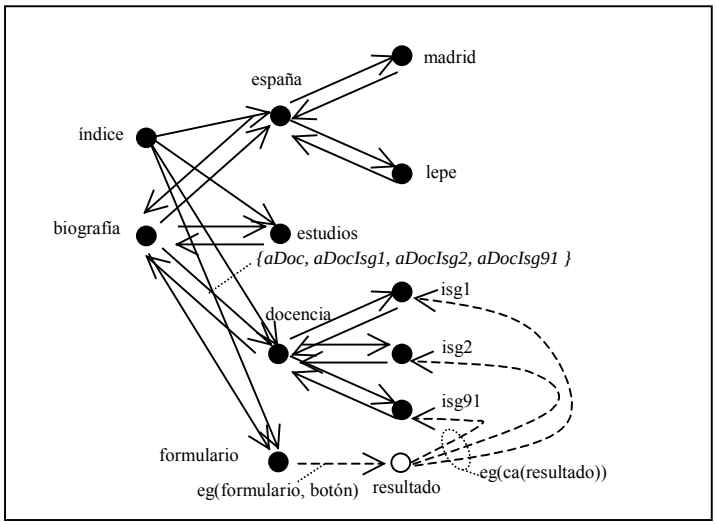

Figura 5.30 Grafo de contenidos

Si recordamos el esquema navegacional del Capítulo 2, teníamos dos ventanas formada por uno y dos paneles respectivamente. Ampliaremos el ejemplo con otro panel más en la primera ventana para el índice. Por tanto en el primer panel de la primera ventana se muestra el índice. En el segundo panel de esa ventana la biografía. En el primer panel de la segunda ventana se muestra la información proveniente de la primera ventana (españa, estudios, docencia y formulario). En el segundo panel de dicha ventana el resto de información. La Figura 5.31 muestra este esquema navegacional así como su relación con el grafo de contenidos.

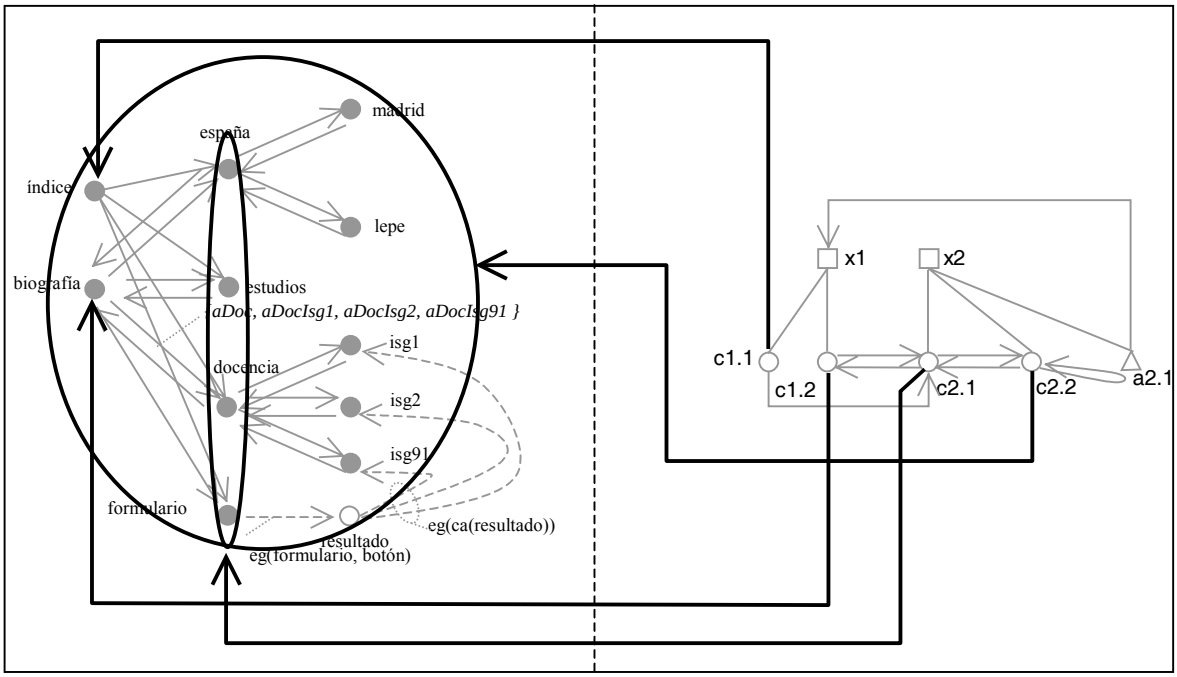

Figura 5.31 Esquema navegacional y relación con el grafo de contenidos. La canalización de enlaces puede deducirse de la asignación de contenidos.

Su representación conjuntista es la siguiente.

N = { x1, c1.1, c1.2, x2, c2.1, c2.2, a2.1 }

A ={ (x1, c1.1, conexión), (x1, c1.2, conexión), (c1.2, c2.1, enlace), (c2.1, c1.2, enlace),

(x2, c2.1, conexión), (x2, c2.2, conexión), (x2, a2.1, conexión), (c2.1, c2.2, enlace), (c2.2, c2.1, enlace), (c2.2, c2.2, enlace), (a2.1, x1, enlace) }

d = { (**c1.1**, índice, ∅), (**c1.2**, biografía, ∅),
(**c2.1**, null, {españa, estudios, docencia, formulario),
(**c2.2**, null, {madrid, lepe, isg1, isg2, isg3, isg91}
∪ {resultado} ∪ cg(ca(resultado)) ∪ C) }

l = { (**c1.1, c2.1, enlace**,{(índice, aEsp, españa, total), (índice, aEst, estudios, total), (índice, aDoc, docencia, total), (índice, aForm, formulario, total) },
(**c1.2, c2.1, enlace**, {(biografía, aEsp, españa, total), (biografía, aEst, estudios, total), (biografía, aDoc, docencia, total), (biografía, aDocIsg1, docencia, destIsg1), (biografía, aDocIsg2, docencia, destIsg2), (biografía, aDocIsg91, docencia, destIsg91), (biografía aForm, formulario, total)}),
(**c2.1, c1.2, enlace**, {(españa, aBio, biografía, total), (estudios, aBio, biografía, total), (docencia, aBio, biografía, total), (formulario, aBio, biografía, total)}),
(**c2.1, c2.2, enlace**, {(españa, aMad, madrid, total), (españa, aLepe, lepe, total), (docencia, aIsg1, isg1, total), (docencia, aIsg2, isg2, total), (docencia, aIsg91, isg91, total)} ∪ eg(formulario, botón)),
(**c2.2, c2.2, enlace**, eg(ca(resultado)) ∪ E) }

Nótese que tal y como indica la Figura 5.27 el nodo c2.2 tiene asignados todos los contenidos.

Ahora veamos como se comporta dinámicamente el modelo.

- Supongamos que partimos de la activación de la primera ventana.

  a(x1) = (Activos1, Mostrar1)

  Activos1 = { x1, c1.1, c1.2 }
  Mostrar1 = { (c.1.1, índice), (c1.2, biografía) }

  Estado que queda recogido en la Figura 5.32.

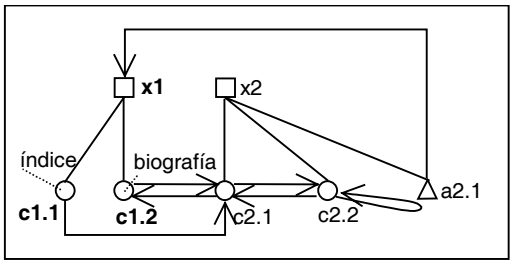

Figura 5.32 Activación de la primera ventana

- Si ahora seleccionamos el enlace de la biografía con los estudios,

  f((biografía, aEst), c1.2, Activos1, Mostrar1)

  ∃2.1((c1.2, c2.1, enlace) ∈ A ∧ enlAct(biografía, aEst) ∈ l(c1.2, c2.1, enlace))
  ∧ (c1.2, biografía) ∈ Mostrar2

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que,

  ∃1∃2 ((x1, c1.2, conexión) ∈ A ∧ (x2, c2.1, conexión) ∈ A ∧ 1≠2)

  Activos2=$\prod_1$a(x2) = {x2, c2.1, c2.2, a2.1}
  Mostrar2= {(c2.1, estudios)} ∪ {(c2.2, null)}

  Ilustrado en la Figura 5.33. Nótese que el contenido null no se ha hecho explícito.

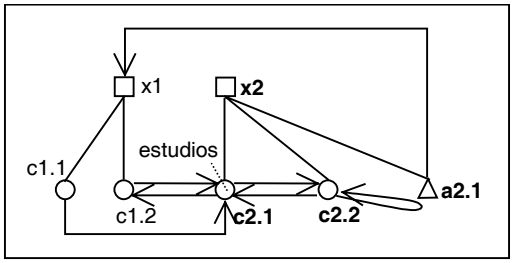

Figura 5.33 Selección de estudios

- Ahora solo podemos seleccionar el enlace con la biografía, o activar el activador de nexo a2.1. Supongamos que activamos el enlace de biografía.

  f((estudios, aBio), c2.1, Activos2, Mostrar2)

  $\exists 1.2((c2.1, c1.2, enlace) \in A \wedge enlAct(estudios, aBio) \in l(c2.1, c1.2, enlace))$
  $\wedge (c2.1, estudios) \in Mostrar3$

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que
  $\exists 2 \exists 1 ((x2, c2.1, conexión) \in A \wedge (x1, c1.2, conexión) \in A \wedge 2 \neq 1)$

  $Activos3 = \prod_1 a(x1) = \{x1, c1.1, c1.2\}$
  $Mostrar3 = \{(c1.2, biografía)\} \cup \{(c1.1, índice)\}$

  Ilustrado en la Figura 5.34.

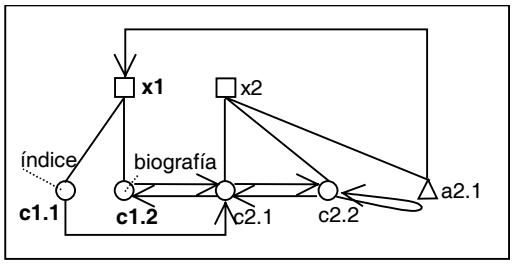

Figura 5.34 Vuelta a biografía

- Supongamos que ahora seleccionamos en el índice el formulario de búsqueda.

  f((índice, aForm), c1.1, Activos3, Mostrar3)

  $\exists 2.1((c1.1, c2.1, enlace) \in A \wedge enlAct(índice, aForm) \in l(c1.1, c2.1, enlace))$
  $\wedge (c1.1, índice) \in Mostrar4$

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que

  $\exists 1 \exists 2 ((x1, c2,1, conexión) \in A \wedge (x2, c2.1, conexión) \in A \wedge 1 \neq 2)$

  $Activos4 = \prod_1 a(x2) = \{x2, c2.1, c2.2, a2.1\}$
  $Mostrar4 = (c2.1, formulario) \cup \{(c2.2, null)\}$

  Ilustrado en la Figura 5.35.

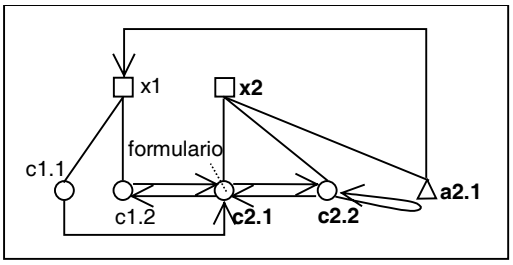

Figura 5.35 Selección del formulario

- Si ahora introducimos en el formulario la palabra UML y pulsamos el botón de búsqueda tenemos que,

  f((formulario, botón), c2.1, Activos5, Mostrar5)

  $\exists 2.2((c2.1, c2.2, enlace) \in A \wedge enlAct(formulario, botón) \in l(c2.1, c2.2, enlace))$
  $\wedge (c2.1, formulario) \in Mostrar5$

  Nótese que el nodo contenedor pertenece al mismo nodo nexo, ya que

  $\exists 2\ ((x2, c2.1, conexión) \in A \wedge (x2, c2.2, conexión) \in A)$

  Activos5 = Activos4
  Mostrar5 = Mostrar4 \ {(c2.2, null)} $\cup$ {(c2.2, conAcc(formulario, botón))}
  = {(c2.1, formulario), (c2.2, resultado)}

  Donde conAcc(formulario, botón) = cg(formulario, botón) = resultado. Podemos ver el resultado en la Figura 5.36.

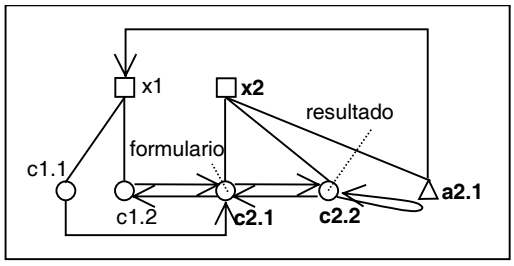

Figura 5.36 Resultado de la búsqueda

- Si ahora seleccionamos el enlace del resultado con isg1 tenemos,

  f((resultado, aIsg1), c2.2, Activos5, Mostrar5)

  $\exists 2.2((c2.2, c2.2, enlace) \in A \wedge enlAct(resultado, aIsg1) \in l(c2.2, c2.2, enlace))$
  $\wedge (c2.2, resultado) \in Mostrar6$

  Activos6 = Activos5
  Mostrar6 = Mostrar5 \ {(c2.2, resultado)} $\cup$ {(c2.2, conAcc(resultado, aIsg1))}
  = {(c2.1, formulario), (c2.2, isg1)}

  Nótese que enlAct(resultado, aIsg1) $\in$ eg(resultado, aIsg1) $\in$ eg(ca(resultado)) $\in$ l(c2.2, c2.2, enlace). Podemos ver el resultado en la Figura 5.37.

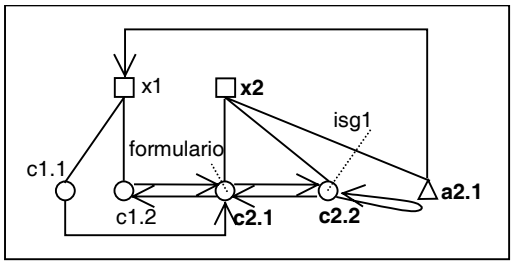

Figura 5.37 Selección de un resultado

- Si ahora seleccionamos el enlace de isg1 con docencia,

  f((isg1, aDoc), c2.2, Activos6, Mostrar6)

  $\exists 2.2((c2.2, c2.2, enlace) \in A \wedge enlAct(isg1, aDoc) \in l(c2.2, c2.2, enlace))$
  $\wedge (c2.2, isg1) \in Mostrar6$

  Activos7 = Activos6
  Mostrar7 = Mostrar6 \ {(c2.2, isg1)} $\cup$ {(c2.2, conAcc(isg1, aDoc))}
  = {(c2.1, formulario), (c2.2, docencia)}

  Nótese que enlAct(isg1, aDoc) = (isg1, aDoc, docencia, total) $\in E \in$ l(c2.2, c2.2, enlace). Podemos ver el resultado en la Figura 5.38.

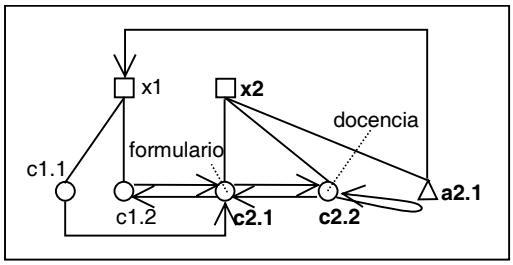

Figura 5.38 Enlace con la docencia

SP: la semántica presentacional del modelo

La semántica presentacional es muy similar a la navegacional pero incluyendo información de cómo van a ser presentados los nodos nexo, contenedor, y sus contenidos. La única modificación se centra en el conjunto Mostrar, que ahora incluye información sobre la función p, así que simplemente nos limitaremos a presentar las diferencias. Primero redefiniremos el conjunto M para que ahora incluya información de presentación. En efecto, ahora tenemos que

$$M = (N_c \times C \times EP \times EPC) \cup (N_c \times C \times \mathbb{R} \times EP \times EPC)$$

Definimos,

$$SN = (a, f)$$

donde:

$$a: N_x \rightarrow \wp(V) \times \wp(M)$$

de tal forma que $a(x_i) = (Activos_n, Mostrar_n)$

donde,

$$\begin{aligned}\text{Activos}_n = \{x_i\} &\cup \{\, c_j \mid (x_i, c_j, \text{conexión}) \in A \,\} \\ &\cup \{\, a_j \mid (x_i, a_j, \text{conexión}) \in A \,\} \\ &\cup \{\, c_j \mid (x_i, c_j, \text{conexiónS}, t) \in A \,\} \\ &\cup \{\, (x_j, t) \mid (x_i, x_j, \text{enlaceS}, t) \in A \,\}\end{aligned}$$

$$\begin{aligned}\text{Mostrar}_n = &\{\, (c_j, d_1(c_j), p(c_j), p(d_1(c_j))) \mid (x_i, c_j, \text{conexión}) \in A \,\} \\ \cup &\{\, (c_j, d_1(c_j), t, p(c_j), p(d_1(c_j))) \mid (x_i, c_j, \text{conexiónS}, t) \in A \,\}\end{aligned}$$

Redefinimos la función de activación de enlaces f,

$$f: ((C_{ns} \times A) \cup \{\perp\}) \times N \times \wp(V) \times \wp(M) \rightarrow \wp(V) \times \wp(M)$$

Donde el símbolo $\perp$ se utiliza para los enlaces que no tienen origen en enlaces de contenido (es decir los sincro entre nodos nexo, y los activadores de nexos). Veamos que sucede con f según la naturaleza del nodo en que se origine el enlace, y el nodo destino del enlace.

- El origen se produce en un nodo de contenidos, y el destino es otro nodo de contenidos, es decir,

  $f((o, a), c_i, \text{Activos}_n, \text{Mostrar}_n) = (\text{Activos}_{n+1}, \text{Mostrar}_{n+1})$

  y $\exists j\ ((c_i, c_j, \text{enlace}) \in A \wedge \text{enlAct}(o, a) \in l(c_i, c_j, \text{enlace})) \wedge (c_i, o) \in \text{Mostrar}_n$

  - Si $\exists r \exists s((x_r, c_i, \text{conexión}) \in A \wedge (x_s, c_j, \text{conexión}) \in A \wedge r \neq s)$.

    $\text{Activos}_{n+1} = \prod_1 a(x_s)$

    $$\begin{aligned}\text{Mostrar}_{n+1} = &\{(c_j, \text{conAcc}(o, a), p(c_j), p(\text{conAcc}(o, a)))\} \\ \cup &\{(c_k, d_1(c_k), P(c_k), p(d_1(c_k))) \mid (x_s, c_k, \text{conexión}) \in A \wedge k \neq j\} \\ \cup &\{(c_k, d_1(c_k), t, p(c_k), p(d_1(c_k)) \mid (x_s, c_k, \text{conexiónS}, t) \in A \wedge k \neq j\}\end{aligned}$$

  - Si $\exists r((x_r, c_i, \text{conexión}) \in A \wedge (x_r, c_j, \text{conexión}) \in A)$.

    $$\text{Activos}_{n+1} = \text{Activos}_n$$

    $$\begin{aligned}\text{Mostrar}_{n+1} = \text{Mostrar}_n \setminus &\{(c_j, m, p_1, p_2) \mid (c_j, m, p_1, p_2) \in \text{Mostrar}_n\} \\ &\cup \{(c_j, \text{conAcc}(o, a), p(c_j), p(\text{conAcc}(o, a)))\}\end{aligned}$$

    Es decir, se activa el nodo contenedor y se muestra el contenido seleccionado por el enlace.

- El origen se produce en un nodo nexo, y por tanto es un enlace sincro. Nótese que en este caso no tenemos enlace de contenidos y usamos $\perp$.

  $f(\perp, x_i, \text{Activos}_n, \text{Mostrar}_n) = a(x_j)$
  si $((x_j, t) \in \text{Activos}_n \wedge t = \text{tiempo}(x_i))$

  Donde la función tiempo(x_i) devuelve el tiempo transcurrido desde la activación del nodo nexo x_i.
  Nótese que es necesario que solo parta un enlace sincro del nodo origen, y que nos limitamos a activar el nodo destino, cuando han transcurrido t segundos desde la activación de x_i.

- El origen se produce en un activador de nexos. Nótese que en este caso no tenemos enlace de contenidos y usamos.

  $f(\perp, a_i, \text{Activos}_n, \text{Mostrar}_n) = a(x_j)$ si $\exists j\ ((a_i, x_j, \text{enlace}) \in A) \wedge a_i \in \text{Mostrar}_n$

En este caso simplemente activamos el nodo destino. Nótese que por como hemos definido A, los activadores de nexos solo pueden enlazar con un destino.

- En otro caso, f se comporta como la identidad.

Esta semántica de presentación no trata de cubrir todas las posibilidades soportadas por una interfaz gráfica de usuario moderna (al igual que la función de presentación p, sobre la que está definida). La razón de este hecho se debe a utilizar Pipe como herramienta de conceptualización-prototipado. En caso de querer utilizarlo para las fases de diseño-construcción, y por tanto como metodología, sería necesaria una ampliación de la función de presentación p, y por extensión de esta semántica.

5.3 Ejemplo

Vamos a ver un ejemplo de cómo el modelo Pipe es capaz de representar una aplicación hipermedia. Retomaremos el ejemplo visto en el capítulo de los sistemas de representación hipermedia y a lo largo de este capítulo, ampliando sus posibilidades de esquema navegacional. Aunque exista información duplicada respecto al ejemplo del apartado anterior, hemos preferido repetirla para hacer este punto autocontenido. En efecto, supongamos que queremos modelar los contenidos de la página web de un profesor de universidad (para centrar ideas elegiremos al autor de estos párrafos). En dichos contenidos aparecerá un texto inicial que contendrá una breve biografía del profesor. Esta biografía enlaza con un mapa España, país de nacimiento del autor, y viceversa (es decir, el mapa de España también presenta un enlace con la biografía). Dentro de este mapa tenemos enlaces con descripciones de diversas ciudades de la geografía española que son de interés para el profesor, y viceversa. En particular destacan las ciudades de Madrid, y la localidad de Lepe. La biografía también presenta un enlace a una descripción los estudios cursados por el profesor, y viceversa. También enlaza con una descripción de la docencia que imparte el profesor y viceversa. Además, la biografía permite enlazar con descripciones concretas a cada asignatura dentro de la descripción genérica. La descripción de asignaturas enlaza con el programa de cada una de ellas y viceversa. Además hay un índice representado por una imagen que permite acceder al país de nacimiento, a los estudios y a la docencia. También hay un formulario de búsqueda de contenidos, accesible desde el índice y la biografía (y viceversa).

Como ya vimos en el apartado sobre contenidos y relaciones tenemos que:

C_A = { biografía, índice, españa, madrid, lepe, estudios, docencia, isg1, isg2, isg91, formulario, resultado }
∪ {imagen, audio}

E_A = { (biografía, aEsp, españa, total), (españa, aBio, biografía, total),
(españa, aMad, Madrid, total), (madrid, aEsp, españa, total), (españa, aLepe, lepe, total),
(lepe, aEsp, españa, total),
(biografía, aEst, estudios, total), (estudios, aBio, biografía, total),
(biografía, aDoc, docencia, total), (docencia, aBio, biografía, total),
(biografía, aDocIsg1, docencia, isg1), (biografía, aDocIsg2, docencia, isg2),
(biografía, aDocIsg91, docencia, isg91),
(docencia, aIsg1, isg1, total), (isg1, aDoc, docencia, total),
(docencia, aIsg2, isg2, total), (isg2, aDoc, docencia, total),
(docencia, aIsg91, isg91, total), (isg91, aDoc, docencia, total),
(biografía, aForm, formulario, total), (formulario, aBio, biografía, total),
(índice, aEsp, españa, total), (índice, aEst, estudios, total), (índice, aDoc, docencia, total) }
∪ eg(formulario, botón)) ∪ eg(ca(resultado)))

Es decir nos encontramos con la situación descrita en la Figura 5.39, similar a la Figura 5.9, y supuesto que concretamos los eg(ca(resultado)). En esta figura hemos optado por eliminar la información sobre las anclas.

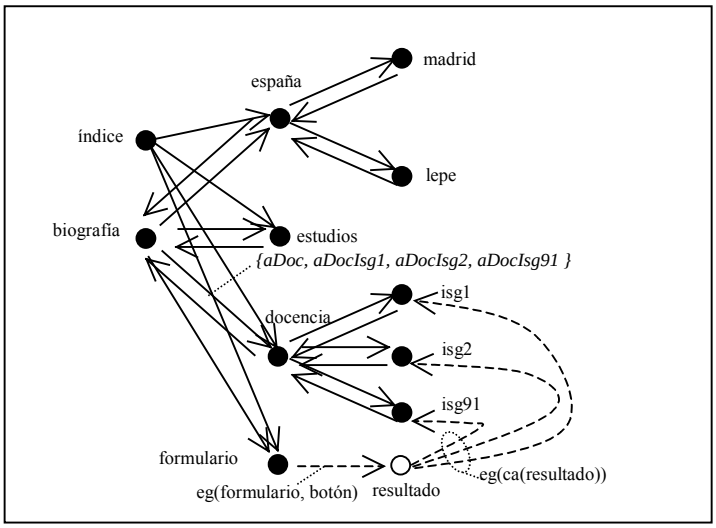

Figura 5.39 Grafo de contenidos

Ahora vamos a considerar el siguiente esquema navegacional (obviaremos la información presentacional). Hay una ventana inicial en la que aparece un logotipo y suena una música de fondo. Si tras cinco segundos no se selecciona ningún botón se procederá con el primer esquema navegacional. Además, la ventana inicial contiene cuatro botones que dan acceso a otros cuatro esquemas distintos. El primer esquema (por defecto) consiste en una única ventana en la que se muestran todos los contenidos. El segundo está formado por una ventana inicial en la que se muestran el índice y la biografía. Ambos dan acceso a otra ventana con dos paneles. En uno se muestra lo seleccionado en la ventana anterior (españa, estudios, docencia o formulario), y en el otro se muestra lo seleccionado en el primer panel. También hay un botón que da acceso a la primera ventana. En el tercer esquema navegacional se consideran cuatro ventanas. En la primera se muestra la biografía y el índice, en la segunda se muestra lo seleccionado en la primera ventana (españa, estudios, docencia o formulario). En la tercera se muestra lo seleccionado en la segunda (ciudades, asignaturas, o el resultado de la búsqueda). Finalmente, en la cuarta ventana se muestran los contenidos que enlazan con los resultados de la búsqueda. También hay un botón que da acceso a la primera ventana. En el cuarto esquema navegacional tenemos dos ventanas con dos paneles cada una. En los dos primeros paneles se muestra el índice y la biografía. En el tercero los enlaces que provienen de la primera ventana. En el cuarto los enlaces que provienen de la tercera ventana. Finalmente en la quinta opción tenemos una ventana con seis paneles. Uno para el índice, otro para la biografía, otro para españa y los estudios, otro para docencia, otro para el formulario de búsqueda y el último para los enlaces procedentes de los cuatro últimos paneles, y para los enlaces del formulario de búsqueda. Es decir, el esquema navegacional queda recogido en la Figura 5.40.

Nótese que no tiene sentido conectar paneles de la misma ventana en ambas direcciones salvo que sea posible acceder a ambas paneles desde otras ventanas. De esta forma, por ejemplo si hay un enlace en ambas direcciones entre los nodos c8.2 y c8.3, pero solamente es simple entre c3.1 y c3.2. Esto permitirá que cuando acceda a c8.3 desde c9.1 se puedan mostrar los contenidos de los enlaces canalizados a través de (c8.3, c8.2, enlace). Como no es posible acceder a c3.2 desde una ventana exterior, el enlace no es necesario.

La representación conjuntista de la Figura 5.36 viene dada por los siguientes conjuntos. Para facilitar la legibilidad hemos decidido agruparlos por esquemas interfaces concretas, aunque evidentemente esto no es necesario.

N = { x0, c0.1, c0.2, a0.1, a0.2, a0.3, a0.4 }
∪ { x1, c1.1}
∪ { x2, c2.1, c2.2, x3, c3.1, c3.2, a3.1 }
∪ { x4, c4.1, c4.2, x5, c5.1, x6, c6.1, x7, c7.1, a7.1 }
∪ { x8, c8.1, c8.2, c8.3, x9, c9.1, c9.2, a9.1}
∪ { x10, c10.1, c10.2, c10.3, c10.4, c10.5, c10.6 }

A = { (x0, c0.1, conexión), (x0, c0.2, conexión), (x0, a0.1, conexión), (x0, a0.2, conexión), (x0, a0.3, conexión), (x0, a0.4, conexión), (x0, x1, enlaceS, 5) }

∪ { (x1, c1.1, conexión), (c1.1, c1.1, enlace) }
∪ { (a0.1, x2, enlace),
(x2, c2.1, conexión), (x2, c2.2, conexión), (c2.2, c3.1, enlace), (c3.1, c2.2, enlace),
(x3, c3.1, conexión), (x3, c3.2, conexión), (x3, a3.1, conexión), (c3.1, c3.2, enlace), (c3.2, c3.2, enlace),
(a3.1, x2, enlace) }
∪ { (a0.2, x4, enlace),
(x4, c4.1, conexión), (x4, c4.2, conexión), (c4.1, c5.1, enlace), (c4.2, c5.1, enlace), (c5.1, c4.2, enlace),
(x5, c5.1, conexión), (c5.1, c6.1, enlace), (c6.1, c5.1, enlace),
(x6, c6.1, conexión), (c6.1, c7.1, enlace), (x7, c7.1, conexión), (c7.1, c7.1, enlace),
(x7, a7.1, conexión), (a7.1, x4, enlace) }
∪ { (a0.3, x8, enlace),
(x8, c8.1, conexión), (x8, c8.2, conexión), (x8, c8.3, conexión), (c8.1, c8.3, enlace), (c8.2, c8.3, enlace),
(c8.3, c8.2, enlace), (c8.3, c9.1, enlace), (c9.1, c8.3, enlace),
(x9, c9.1, conexión), (x9, c9.2, conexión), (x9, a9.1, conexión), (a9.1, x8, enlace), (c9.1, c9.2, enlace),
(c9.2, c9.2, enlace) }
∪ { (a0.4, x10, enlace),
(x10, c10.1, conexión), (x10, c10.2, conexión), (x10, c10.3, conexión), (x10, c10.4, conexión),
(x10, c10.5, conexión), (x10, c10.6, conexión), (c10.1, c10.3, enlace),
(c10.1, c10.4, enlace), (c10.1, c10.5, enlace), (c10.2, c10.3, enlace),
(c10.2, c10.4, enlace), (c10.2, c10.5, enlace), (c10.3, c10.6, enlace),
(c10.4, c10.6, enlace), (c10.5, c10.6, enlace), (c10.6, c10.6, enlace) }

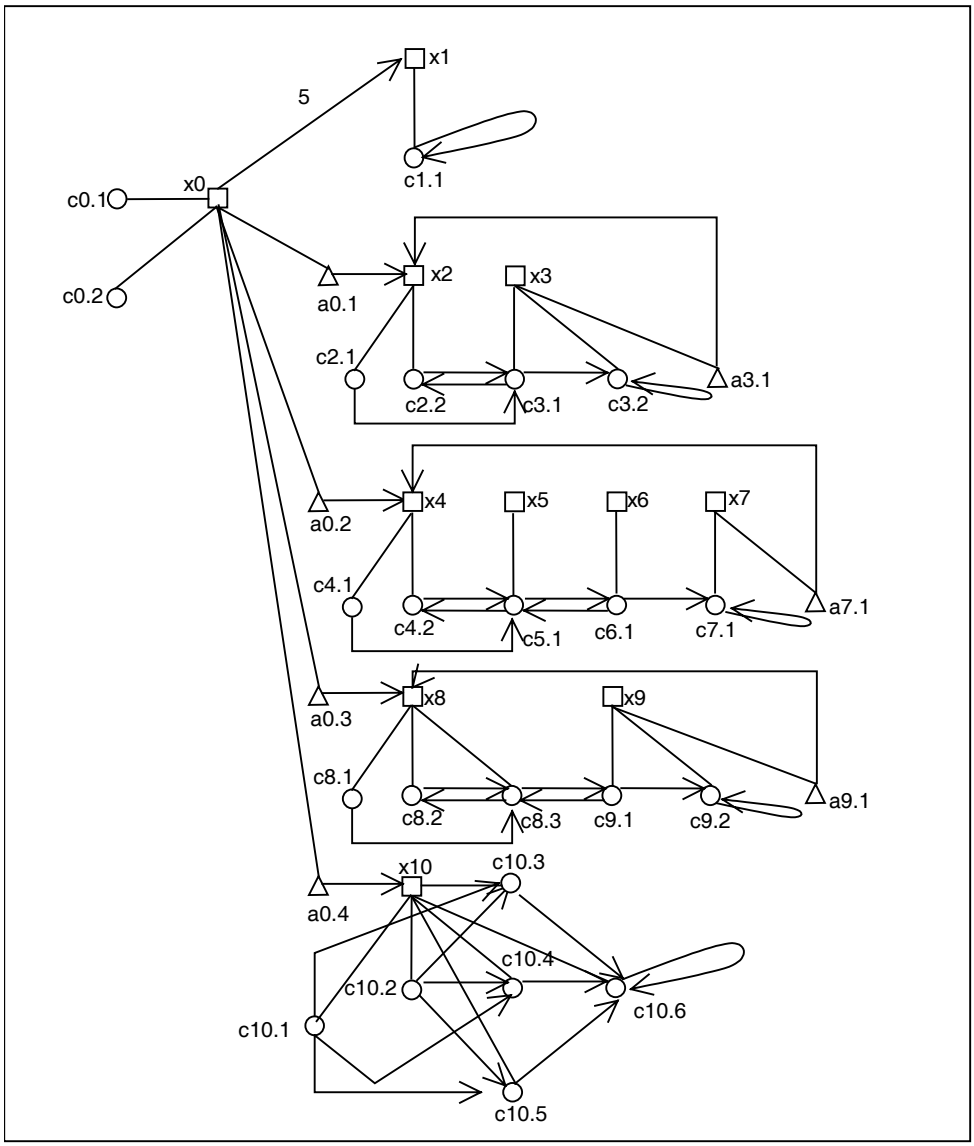

Figura 5.40 Esquema navegacional de la aplicación

La función de asignación de contenidos es la siguiente, supuesto que aumentamos los contenidos con imagen y audio (los nodos en negrita):

d = { (**c0.1**, imagen, ∅), (**c0.2**, audio, ∅) }
∪ { (**c1.1**, biografía, {españa, madrid, lepe, estudios, docencia, isg1, isg2, isg91, formulario}
∪ {resultado} ∪ cg(ca(resultado)) ∪ C) }
∪ { (**c2.1**, índice, ∅), (**c2.2**, biografía, ∅),
(**c3.1**, null, {españa, estudios, docencia, formulario),
(**c3.2**, null, {madrid, lepe, isg1, isg2, isg3, isg91}
∪ {resultado} ∪ cg(ca(resultado)) ∪ C) }
∪ { (**c4.1**, índice, ∅), (**c4.2**, biografía, ∅),
(**c5.1**, null, {españa, estudios, docencia, formulario),
(**c6.1**, null, {madrid, lepe, isg1, isg2, isg3, isg91} ∪ {resultado}),
(**c7.1**, null, cg(ca(resultado)) ∪ C) }
∪ { (**c8.1**, índice, ∅), (**c8.2**, biografía, ∅), (**c8.3**, null, {españa, estudios, docencia, formulario),
(**c9.1**, null, {madrid, lepe, isg1, isg2, isg3, isg91} ∪ {resultado}),
(**c9.2**, null, cg(ca(resultado)) ∪ C), }
∪ { (**c10.1**, índice, ∅), (**c10.2**., biografía, ∅), (**c10.3**, null, {españa, estudios}), (**c10.4**, null, {docencia}),
(**c10.5**, null, {formulario}), (**c10.6**, null, {madrid, lepe, isg1, isg2, isg91}
∪ {resultado} ∪ cg(ca(resultado)) ∪ C) }

Finalmente la función de canalización de enlaces es (las tuberías en negrita):

l = { (**c1.1, c1.1, enlace**, E^0 ∪ eg(formulario, botón) ∪ eg(ca(resultado)) ∪ E),
(**c2.1, c3.1, enlace**,{(índice, aEsp, españa, total), (índice, aEst, estudios, total),
(índice, aDoc, docencia, total), (índice, aForm, formulario, total)}),
(**c2.2, c3.1, enlace**, {(biografía, aEsp, españa, total), (biografía, aEst, estudios, total),
(biografía, aDoc, docencia, total), (biografía, aDocIsg1, docencia, destIsg1),
(biografía, aDocIsg2, docencia, destIsg2), (biografía, aDocIsg91, docencia, destIsg91),
(biografía aForm, formulario, total)}),
(**c3.1, c2.2, enlace**, {(españa, aBio, biografía, total), (estudios, aBio, biografía, total),
(docencia, aBio, biografía, total), (formulario, aBio, biografía, total)}),
(**c3.1, c3.2, enlace**, {(españa, aMad, madrid, total), (españa, aLepe, lepe, total),
(docencia, aIsg1, isg1, total), (docencia, aIsg2, isg2, total),
(docencia, aIsg91, isg91, total)} ∪ eg(formulario, botón)),
(**c3.2, c3.2, enlace**, eg(ca(resultado)) ∪ E),
(**c4.1, c5.1, enlace**, {(índice, aEsp, españa, total), (índice, aEst, estudios, total),
(índice, aDoc, docencia, total), (índice, aForm, formulario, total) }),
(**c4.2, c5.1, enlace**, {(biografía, aEsp, españa, total), (biografía, aEst, estudios, total),
(biografía, aDoc, docencia, total), (biografía, aDocIsg1, docencia, destIsg1),
(biografía, aDocIsg2, docencia, destIsg2), (biografía, aDocIsg91, docencia, destIsg91),
(biografía aForm, formulario, total)}),
(**c5.1, c4.2, enlace**, {(españa, aBio, biografía, total), (estudios, aBio, biografía, total),
(docencia, aBio, biografía, total), (formulario, aBio, biografía, total)}),
(**c5.1, c6.1, enlace**, {(españa, aMad, madrid, total), (españa, aLepe, lepe, total),
(docencia, aIsg1, isg1, total), (docencia, aIsg2, isg2, total),
(docencia, aIsg91, isg91, total)} ∪ eg(formulario, botón)),
(**c6.1, c5.1, enlace**, {(madrid, aEsp, españa, total), (lepe, aEsp, españa, total), (isg1, aDoc, docencia),
(isg2, aDoc, docencia, total), (isg91, aDoc, docencia, total) }),
(**c6.1, c7.1, enlace**, eg(ca(resultado))),
(**c7.1, c7.1, enlace**, E)
(**c8.1, c8.3, enlace**, {(índice, aEsp, españa, total), (índice, aEst, estudios, total),
(índice, aDoc, docencia, total), (índice, aForm, formulario, total) }),
(**c8.2, c8.3, enlace**, {(biografía, aEsp, españa, total), (biografía, aEst, estudios, total),
(biografía, aDoc, docencia, total), (biografía, aDocIsg1, docencia, destIsg1),
(biografía, aDocIsg2, docencia, destIsg2), (biografía, aDocIsg91, docencia, destIsg91),
(biografía aForm, formulario, total)}),
(**c8.3, c8.2, enlace**, {(españa, aBio, biografía, total), (estudios, aBio, biografía, total),
(docencia, aBio, biografía, total), (formulario, aBio, biografía, total)}),
(**c8.3, c9.1, enlace**, {(españa, aMad, madrid, total), (españa, aLepe, lepe, total),
(docencia, aIsg1, isg1, total), (docencia, aIsg2, isg2, total),
(docencia, aIsg91, isg91, total)} ∪ eg(formulario, botón)),
(**c9.1, c8.3, enlace**, {(madrid, aEsp, españa, total), (lepe, aEsp, españa, total), (isg1, aDoc, docencia),

(isg2, aDoc, docencia, total), (isg91, aDoc, docencia, total) }),
(**c9.1, c9.2, enlace**, eg(ca(resultado))),
(**c9.2, c9.2, enlace**, E),
(**c10.1, c10.3, enlace**, {(índice, aEsp, españa, total), (índice, aEst, estudios, total)}),
(**c10.1, c10.4, enlace**, {(índice, aDoc, docencia, total)}),
(**c10.1, c10.5, enlace**, {(índice, aForm, formulario, total)}),
(**c10.2, c10.3, enlace**, {(biografía, aEsp, españa, total), (biografía, aEst, estudios, total) }),
(**c10.2, c10.4, enlace,** {(biografía, aDoc, docencia, total), (biografía, aDocIsg1, docencia, destIsg1),
(biografía, aDocIsg2, docencia, destIsg2),
(biografía, aDocIsg91, docencia, destIsg91)}),
(**c10.2, c10.5, enlace**, {(biografía, aForm, formulario, total)}),
(**c10.3, c10.6, enlace**, {(españa, aMad, madrid, total), (españa, aLepe, lepe, total)}),
(**c10.4, c10.6, enlace**, {(docencia, aIsg1, isg1, total), (docencia, aIsg2, isg2, total),
(docencia, aIsg91, isg91, total)}),
(**c10.5, c10.6, enlace**, eg(formulario, botón)),
(**c10.6, c10.6, enlace**, eg(ca(resultado)) ∪ E) }

En este ejemplo se puede comprobar como la función l no es inyectiva. Este hecho se debe a que en el ejemplo tenemos distintas interfaces para los mismos contenidos. Evidentemente no se va a pedir a un diseñador que proporcione manualmente esta información. La forma ideal de especificar esta información es a través de una herramienta CASE, o en su defecto, mediante gráficos como el recogido por la Figura 5.41.

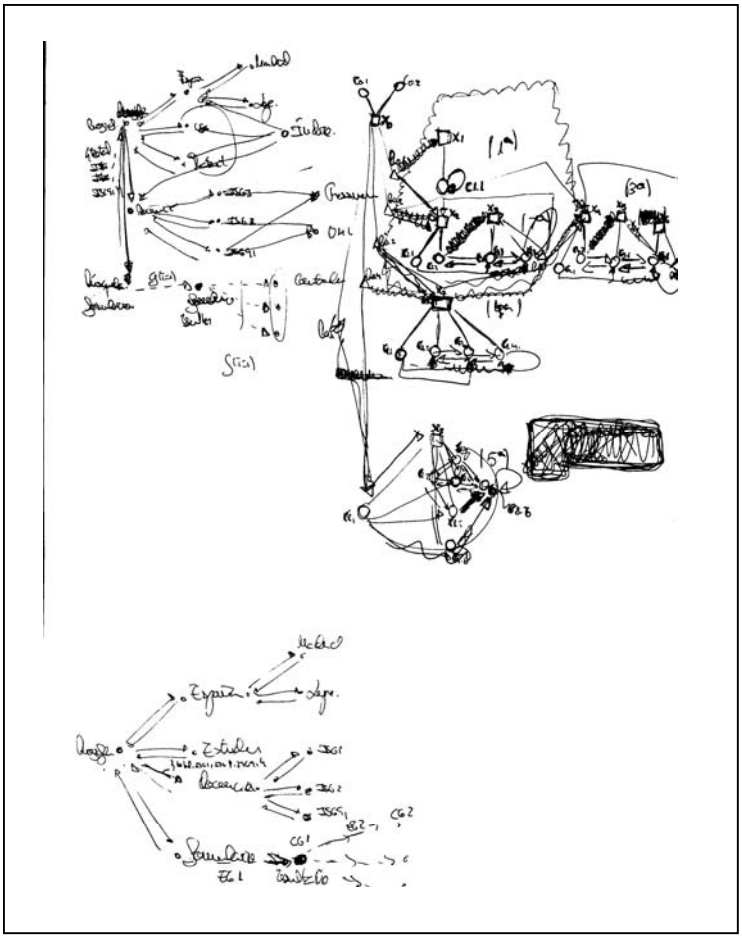

Figura 5.41 ¡La alternativa a las herramientas CASE!

Ahora veamos como se comporta dinámicamente el modelo.

- Iniciemos la interacción por la ventana cero,

a(x0) = (Activos1, Mostrar1)

Activos1 = {x0} ∪ {c0.1, c0.2} ∪ {a0.1, a0.2, a0.3, a0.4} ∪{(x1, 5)}
Mostrar1 = { (c0.1, imagen), (c0.2, audio) }

Gráficamente, podemos observar la activación en la Figura 5.42. Además en la Figura 6.13 del Capítulo 6 podemos ver la aplicación correspondiente a la representación Pipe.

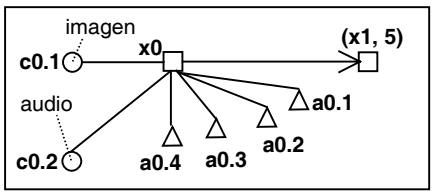

Figura 5.42 Visualización de la ventana inicial

- Pasados los cinco segundos se activa la ventana por defecto,

  f(⊥, x0, Activos1, Mostrar 1) = a(x1)

  ((x1, 5)∈ Activos1 ∧ tiempo() = 5) (han transcurrido 5 segundos desde la activación de x0).

  Activos2 = {x1} ∪ {c1.1}
  Mostrar2= { (c1.1, biografía)}

  Es decir, se muestra la biografía en el nodo c1.1. Lo podemos ver en la Figura 5.43. Además en la Figura 6.15 del Capítulo 6 podemos ver la aplicación correspondiente a la representación Pipe.

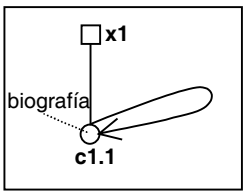

Figura 5.43 Activación de la ventana

- Activamos un enlace de la biografía con un contenido, por ejemplo españa,

  f((biografía, aEsp), c1.1, Activos2, Mostrar2})

  ∃ 1 ((c1.1, c1.1, enlace)∈ A ∧ enlAct(biografía, aEsp)∈ l(c1.1, c1.1, enlace))
  ∧ (c1.1, biografía)∈ Mostrar2

  ∃1 (x1, c1.1, enlace) ∈ A, es decir, el nodo destino está conectado al mismo nodo nexo que el origen (de hecho coinciden).

  Activos3 = Activos2
  Mostrar3 = {(c1.1, biografía)} \ {(c1.1, biografía)}
  ∪ {(c1.1, conAcc(biografía, aEsp))}
  = {(c1.1, españa)}

  Nótese como en el nodo destino de la tubería se muestran los contenidos destino del enlace que se generó en el nodo c1.1 (en este caso españa). La Figura 5.44 ilustra esta situación.

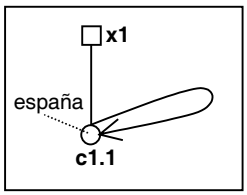

Figura 5.44 Selección de españa

- Nótese que ahora solo podemos activar los enlaces con origen en la foto de **españa**, pues es el único contenido que se está mostrando. Volvamos ahora a la **biografía**, tal y como ilustra la Figura 5.45.

  f((españa, aBio), c1.1, Activos3, Mostrar3)

  $\exists$ 1 ((c1.1, c1.1, enlace)$\in$ A $\wedge$ enlAct(españa, aBio)$\in$ l(c1.1, c1.1, enlace))
  $\wedge$ (c1.1, españa)$\in$ Mostrar3

  $\exists$1 (x1, c1.1, enlace) $\in$ A

  Activos4 = Activos3
  Mostrar4 = {(c1.1, españa)} \ {(c1.1, españa)} $\cup$ {(c1.1, conAcc(españa, aBio))}
  = {(c1.1, biografía)}

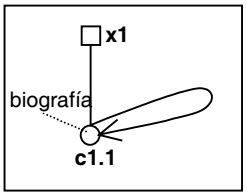

Figura 5.45 Vuelta a la biografía

- Supongamos que ahora activamos el enlace con el **formulario** de búsqueda

  f((biografía, aForm), c1.1, Activos4, Mostrar4)

  $\exists$ 1 ((c1.1, c1.1, enlace)$\in$ A $\wedge$ enlAct(biografía, aForm)$\in$ l(c1.1, c1.1, enlace))
  $\wedge$ (c1.1, biografía)$\in$ Mostrar4

  $\exists$1 (x1, c1.1, enlace) $\in$ A

  Activos5 = Activos4
  Mostrar5 = {(c1.1, biografía)} \ {(c1.1, biografía)}
  $\cup$ {(c1.1, conAcc(biografía, aForm)}
  = { (c1.1, formulario) }

  Es decir, se muestra el destino del enlace. La Figura 5.46 lo ilustra.

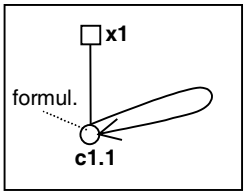

Figura 5.46 Selección del formulario de búsqueda

- Si ahora introducimos en el formulario de búsqueda la palabra UML, y activamos el **botón**

f((formulario, botón), c1.1, Activos5, Mostrar5)

$\exists$ 1 ((c1.1, c1.1, enlace)$\in$ A $\wedge$ enlAct(formulario, botón)$\in$ l(c1.1, c1.1, enlace))
$\wedge$ (c1.1, formulario)$\in$ Mostrar5

$\exists$1 (x1, c1.1, enlace) $\in$ A

Activos6 = Activos5
Mostrar6 = {(c1.1, biografía)} \ {(c1.1, biografía)}
$\cup$ {(c1.1, conAcc(formulario, botón)}
= { (c1.1, resultado) }

Supuesto que al contenido generado tras la búsqueda lo denominamos resultado, es decir, resultado = cg(formulario, botón) = conAcc(formulario, botón). La Figura 5.47 muestra esta situación. Nótese como estamos mostrando contenidos generados dinámicamente.

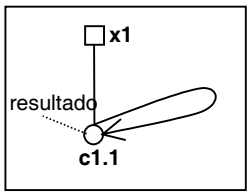

Figura 5.47 Página de resultado de la búsqueda

- Supongamos que la página de resultados enlaza con las páginas de isg1, isg2, isg91 (como vimos para los contenidos en la Figura 5.35). Si ahora seleccionamos el enlace con isg2,

f((resultado, aIsg2), c1.1, Activos6, Mostrar6)

$\exists$ 1 ((c1.1, c1.1, enlace)$\in$ A $\wedge$ enlAct(resultado, aIsg2) $\in$ l(c1.1, c1.1, enlace))
$\wedge$ (c1.1, resultado)$\in$ Mostrar6

$\exists$1 (x1, c1.1, enlace) $\in$ A

Activos7 = Activos6
Mostrar7 = {(c1.1, resultado)} \ {(c1.1, resultado)}
$\cup$ {(c1.1, conAcc(resultado, aIsg2)}
= { (c1.1, isg2) }

Nótese que enlAct(resultado, aIsg2) $\in$ eg(resultado, aIsg2) $\in$ eg(ca(resultado)) $\in$ l(c1.1, c1.1, enlace). El resultado puede verse en la Figura 5.48.

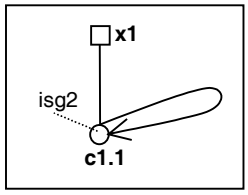

Figura 5.48 Activación de un enlace en el resultado generado

- Si ahora seleccionamos el enlace con la docencia.

f((isg2, aDoc), c1.1, Activos7, Mostrar7)

$\exists$ 1 ((c1.1, c1.1, enlace)$\in$ A $\wedge$ enlAct(isg2, aDoc) $\in$ l(c1.1, c1.1, enlace))
$\wedge$ (c1.1, isg2)$\in$ Mostrar7

$\exists$1 (x1, c1.1, enlace) $\in$ A

Activos8 = Activos7
Mostrar8 = {(c1.1, isg2)} \ {(c1.1, isg2)} ∪ {(c1.1, conAcc(isg2, aDoc)}
= { (c1.1, docencia) }

Nótese que enlAct(isg2, aDoc) = (isg2, aDoc, docencia, total) ∈ E ∈ l(c1.1, c1.1, enlace). El resultado puede verse en la Figura 5.49.

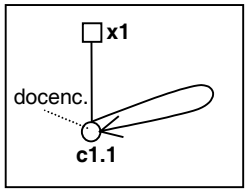

Figura 5.49 Vuelta a la docencia

- Si hubiésemos optado por la activación del activador de nexo a0.1, tendríamos:

  f(⊥, a0.1, Activos1, Mostrar1) = a(x2), ya que

  ∃2 (a0.1, x2, enlace) ∈ A

  Activos2={x2, c2.1, c2.2}
  Mostrar2={(c2.1, índice), (c2.2, biografía)}

  Ilustrado en la Figura 5.50. Además en la Figura 6.17 del Capítulo 6 podemos ver la aplicación correspondiente a la representación Pipe.

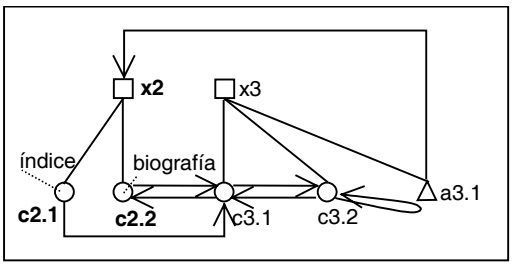

Figura 5.50 Activación de la segunda ventana

- Si ahora seleccionamos el enlace de la biografía con los estudios,

  f((biografía, aEst), c2.2, Activos2, Mostrar2)

  ∃3.1((c2.2, c3.1, enlace) ∈ A ∧ enlAct(biografía, aEst) ∈ l(c2.2, c3.1, enlace))
  ∧ (c2.2, biografía) ∈ Mostrar2

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que,

  ∃2∃3 ((x2, c2.2, conexión) ∈ A ∧ (x3, c3.1, conexión) ∈ A ∧ 2≠3)

  Activos3=$\prod_1$a(x3) = {x3, c3.1, c3.2, a3.1}
  Mostrar3= {(c3.1, estudios)} ∪ {(c3.2, null)}

  Ilustrado en la Figura 5.51. Nótese que el contenido null no se ha hecho explícito.

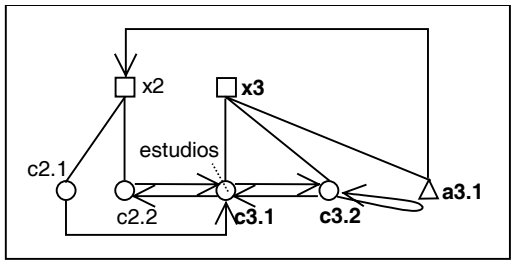

Figura 5.51 Selección de estudios

- Ahora solo podemos seleccionar el enlace con la biografía, o activar el activador de nexo a3.1. Supongamos que activamos el enlace de biografía.

  f((estudios, aBio), c3.1, Activos3, Mostrar3)

  $\exists 2.2((c3.1, c2.2, enlace) \in A \wedge enlAct(estudios, aBio) \in l(c3.1, c2.2, enlace))$
  $\wedge (c3.1, estudios) \in Mostrar3$

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que
  $\exists 3 \exists 2\ ((x3, c3.1, conexión) \in A \wedge (x2, c2.2, conexión) \in A \wedge 3 \neq 2)$

  $Activos4=\prod_1 a(x2) = \{x2, c2.1, c2.2\}$
  $Mostrar4= \{(c2.2, biografía)\} \cup \{(c2.1, índice)\}$

  Ilustrado en la Figura 5.52.

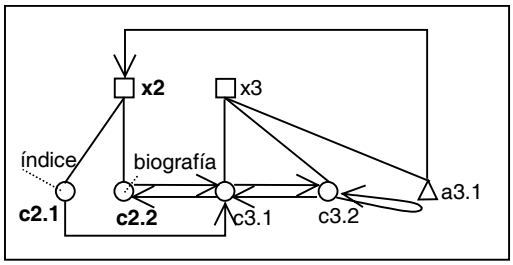

Figura 5.52 Vuelta a biografía

- Supongamos que ahora seleccionamos en el índice el formulario de búsqueda.

  f((índice, aForm), c2.1, Activos4, Mostrar4)

  $\exists 3.1((c2.1, c3.1, enlace) \in A \wedge enlAct(índice, aForm) \in l(c2.1, c3.1, enlace))$
  $\wedge (c2.1, índice) \in Mostrar4$

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que

  $\exists 2 \exists 3\ ((x2, c2,1, conexión) \in A \wedge (x3, c3.1, conexión) \in A \wedge 2 \neq 3)$

  $Activos5=\prod_1 a(x3) = \{x3, c3.1, c3.2, a3.1\}$
  $Mostrar5= (c3.1, formulario) \cup \{(c3.2, null)\}$

  Ilustrado en la Figura 5.53.

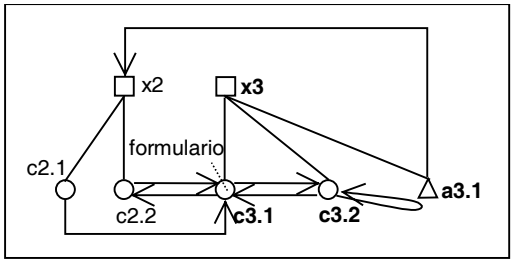

Figura 5.53 Selección del formulario

- Si ahora introducimos en el formulario la palabra UML y pulsamos el botón de búsqueda tenemos que,

  f((formulario, botón), c3.1, Activos5, Mostrar5)

  $\exists 3.2((c3.1, c3.2, enlace) \in A \wedge enlAct(formulario, botón) \in l(c3.1, c3.2, enlace))$
  $\wedge (c3.1, formulario) \in Mostrar5$

  Nótese que el nodo contenedor pertenece al mismo nodo nexo, ya que

  $\exists 3 ((x3, c3.1, conexión) \in A \wedge (x3, c3.2, conexión) \in A)$

  Activos6 = Activos5
  Mostrar6 = Mostrar5 \ {(c3.2, null)} ∪ {(c3.2, conAcc(formulario, botón))}
  = {(c3.1, formulario), (c3.2, resultado)}

  Donde conAcc(formulario, botón) = cg(formulario, botón) = resultado. Podemos ver el resultado en la Figura 5.54.

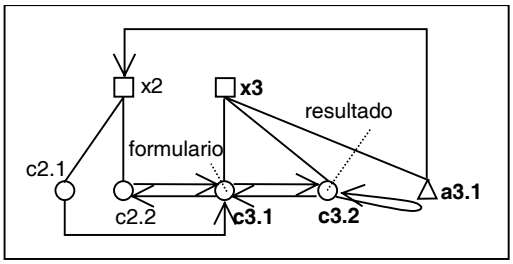

Figura 5.54 Resultado de la búsqueda

- Si ahora seleccionamos el enlace del resultado con isg1 tenemos,

  f((resultado, aIsg1), c3.2, Activos6, Mostrar6)

  $\exists 3.2((c3.2, c3.2, enlace) \in A \wedge enlAct(resultado, aIsg1) \in l(c3.2, c3.2, enlace))$
  $\wedge (c3.2, resultado) \in Mostrar6$

  Activos7 = Activos6
  Mostrar7 = Mostrar6 \ {(c3.2, resultado)} ∪ {(c3.2, conAcc(resultado, aIsg1))}
  = {(c3.1, formulario), (c3.2, isg1)}

  Nótese que enlAct(resultado, aIsg1) ∈ eg(resultado, aIsg1) ∈ eg(ca(resultado)) ∈ l(c3.2, c3.2, enlace). Podemos ver el resultado en la Figura 5.55.

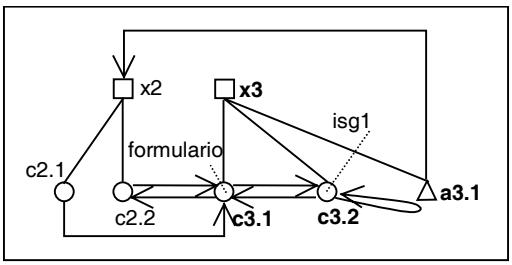

Figura 5.55 Selección de un resultado

- Si ahora seleccionamos el enlace de isg1 con docencia,

f((isg1, aDoc), c3.2, Activos7, Mostrar7)

$\exists 3.2((c3.2, c3.2, enlace) \in A \wedge enlAct(isg1, aDoc) \in l(c3.2, c3.2, enlace))$
$\wedge (c3.2, isg1) \in Mostrar6$

Activos8 = Activos7
Mostrar8 = Mostrar7 \ {(c3.2, isg1)} $\cup$ {(c3.2, conAcc(isg1, aDoc))}
= {(c3.1, formulario), (c3.2, docencia)}

Nótese que enlAct(isg1, aDoc) = (isg1, aDoc, docencia, total) $\in E \in$ l(c3.2, c3.2, enlace). Podemos ver el resultado en la Figura 5.56.

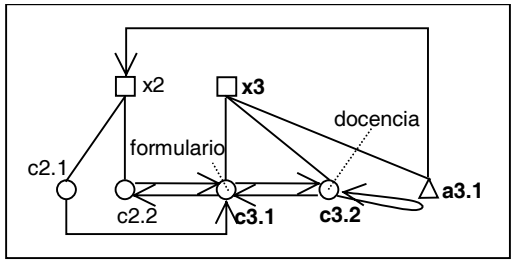

Figura 5.56 Enlace con la docencia

- Si hubiésemos optado por la activación del activador de nexo a0.2, tendríamos:

  f(⊥, a0.2, Activos1, Mostrar1) = a(x4), ya que

  $\exists 2$ (a0.2, x4, enlace) $\in$ A

  Activos2={x4, c4.1, c4.2}
  Mostrar2={(c4.1, índice), (c4.2, biografía)}

  Ilustrado en la Figura 5.57. Además en la Figura 6.19 del Capítulo 6 podemos ver la aplicación correspondiente a la representación Pipe.

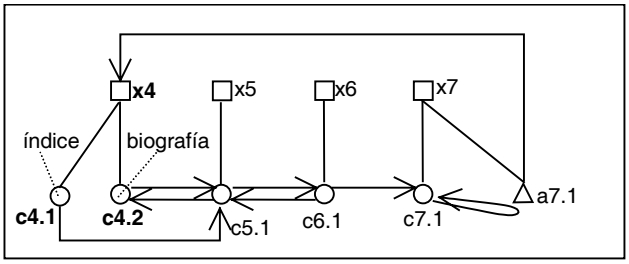

Figura 5.57 Activación de la tercera ventana

- Si ahora activamos el enlace del índice con la foto de españa.

  f((índice, aEsp), c4.1, Activos2, Mostrar2)

  $\exists 5.1((c4.1, c5.1, enlace) \in A \wedge enlAct(índice, aEsp) \in l(c4.1, c5.1, enlace))$
  $\wedge (c4.1, índice) \in Mostrar2$

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que

  $\exists 4 \exists 5$ ((x4, c4,1, conexión) $\in A \wedge$ (x5, c5.1, conexión) $\in A \wedge 4 \neq 5$)

Activos3 = {x5, c5.1}
Mostrar3 = {(c5.1, conAcc(índice, aEsp)} = {(c5.1, españa)}

Podemos ver el resultado en la Figura 5.58.

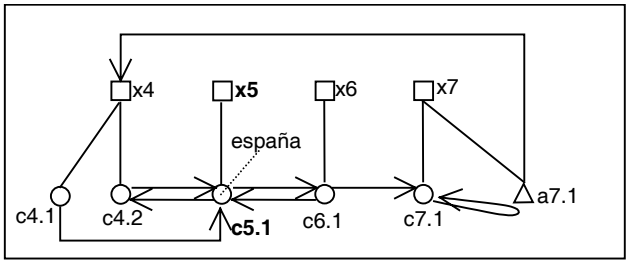

Figura 5.58 Activación de la tercera ventana

- Si volvemos a la biografía,

  f((españa, aBio), c5.1, Activos3, Mostrar3)

  $\exists 5.1((c5.1, c4.2, enlace) \in A \wedge enlAct(españa, aBio) \in l(c5.1, c4.2, enlace))$
  $\wedge (c5.1, españa) \in Mostrar3$

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que

  $\exists 5 \exists 4 ((x5, c5,1, conexión) \in A \wedge (x4, c4.1, conexión) \in A \wedge 5 \neq 4)$

  Activos4 = {x4, c4.1, c4.2}
  Mostrar4 = {(c4.2, biografía)} $\cup$ {(c4.1, índice)}

  Podemos ver el resultado en la Figura 5.59. Nótese que a partir de ahora sustituiremos directamente la expresión conAcc(c, a) por su valor correspondiente.

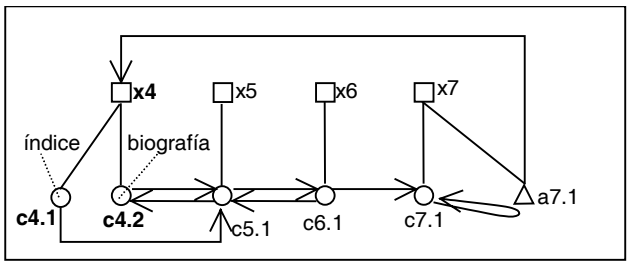

Figura 5.59 Vuelta a la biografía

- Si seleccionamos el enlace de la biografía con el formulario de búsqueda,

  f((biografía, aForm,) c4.2, Activos4, Mostrar4)

  $\exists 5.1((c4.2, c5.1, enlace) \in A \wedge enlAct(biografía, aForm) \in l(c4.2, c5.1, enlace))$
  $\wedge (c4.1, biografía) \in Mostrar4$

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que

  $\exists 4 \exists 5 ((x4, c4.2, conexión) \in A \wedge (x5, c5.1, conexión) \in A \wedge 4 \neq 5)$

  Activos5 = {x5, c5.1}
  Mostrar5 = {(c5.1, formulario)}

  Podemos ver el resultado en la Figura 5.60

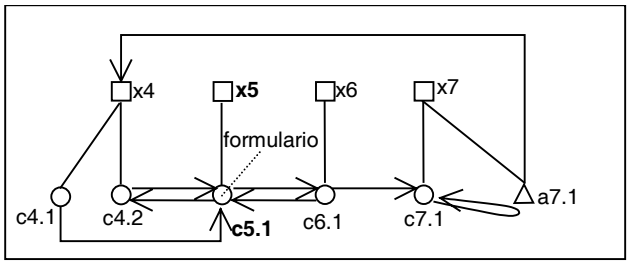

Figura 5.60 Selección del formulario

- Si ahora introducimos en el formulario la palabra UML y buscamos,

  f((formulario, botón), c5.1, Activos5, Mostrar5)

  ∃6.1((c5.1, c6.1, enlace) ∈ A ∧ enlAct(formulario, botón) ∈ l(c5.1, c6.1, enlace))
  ∧ (c5.1, formulario) ∈ Mostrar5

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que

  ∃5∃6 ((x5, c5.1, conexión) ∈ A ∧ (x6, c6.1, conexión) ∈ A ∧ 5≠6)

  Activos6 = {x6, c6.1}
  Mostrar6 = {(c6.1, cg(formulario, botón))} = {(c6.1, resultado)}

  Podemos ver el resultado en la Figura 5.61.

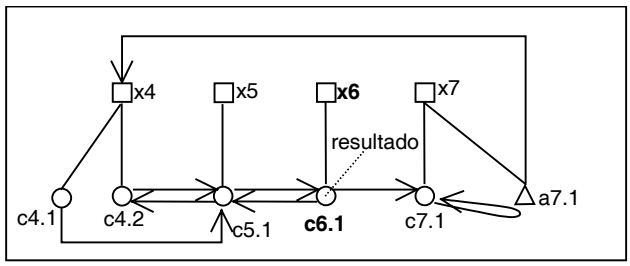

Figura 5.61 Generación del resultado

- Si ahora seleccionamos el enlace con isg2,

  f((resultado, aIsg2), c6.1, Activos6, Mostrar6)

  ∃7.1((c6.1, c7.1, enlace) ∈ A ∧ enlAct(resultado, aIsg2) ∈ l(c6.1, c7.1, enlace))
  ∧ (c6.1, resultado) ∈ Mostrar6

  Nótese que el nodo contenedor pertenece a otro nodo nexo, ya que

  ∃6∃7 ((x6, c6.1, conexión) ∈ A ∧ (x7, c7.1, conexión) ∈ A ∧ 6≠7)

  Activos7 = {x7, c7.1, a7.1}
  Mostrar7 = {(c7.1, isg2)}

Nótese que enlAct(resultado, aIsg2) ∈ eg(resultado, aIsg2) ∈ eg(ca(resultado)) ∈ l(c6.1, c7.1, enlace). Podemos ver el resultado en la Figura 5.62.

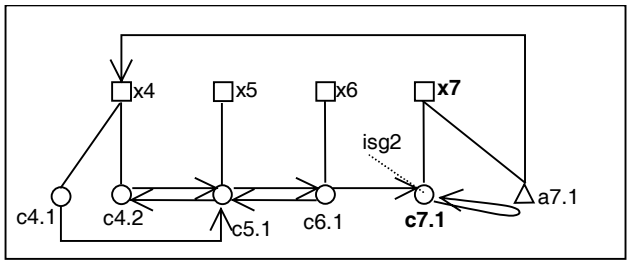

Figura 5.62 Selección de isg2 en resultado

- Si ahora seleccionamos el enlace con docencia,

  f((isg2, aDoc), c7.1, Activos7, Mostrar7)

  ∃7.1((c7.1, c7.1, enlace) ∈ A ∧ enlAct(isg2, aDoc) ∈ l(c7.1, c7.1, enlace))
  ∧ (c7.1, isg2) ∈ Mostrar7

  Activos8 = Activos7
  Mostrar8 = {(c7.1, docencia)}

  Nótese que enlAct(isg2, aDoc) = (isg2, aDoc, docencia, total) ∈ E ∈ l(c7.1, c7.1, enlace). Podemos ver el resultado en la Figura 5.63.

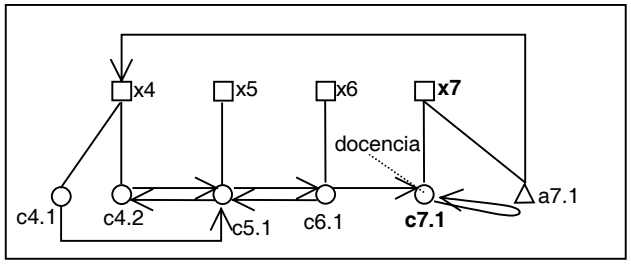

Figura 5.63 Selección de docencia

- Si hubiésemos optado por la activación del activador de nexo a0.3, tendríamos:

  f(⊥, a0.3, Activos1, Mostrar1) = a(x8), ya que

  ∃8 (a0.3, x8, enlace) ∈ A

  Activos2={x8, c8.1, c8.2, c8.3}
  Mostrar2={(c8.1, índice), (c8.2, biografía), (c8.3, null)}

Ilustrado en la Figura 5.64. Además en la Figura 6.21 del Capítulo 6 podemos ver la aplicación correspondiente a la representación Pipe.

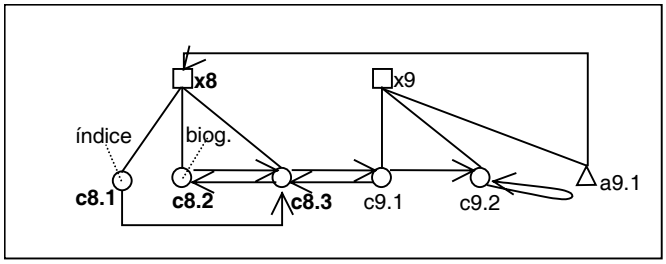

Figura 5.64 Activación de la siguiente ventana

- Si seleccionamos el enlace de la biografía con la docencia, ancla de isg91,

  f((biografía, aDocIsg91), c8.2, Activos2, Mostrar2)

  $\exists 8.3((c8.2, c8.3, enlace) \in A \wedge enlAct(biografía, aDocIsg91) \in l(c8.2, c8.3, enlace))$
  $\wedge (c8.2, biografía) \in Mostrar2$

  $\exists 8 ((x8, c8.2, conexión) \in A \wedge (x8, c8.3, conexión) \in A)$

  Activos3 = Activos2
  Mostrar3 = Mostrar2 \ {(c8.3, null)} $\cup$ {(c8.3, docencia)}

Ilustrado en la Figura 5.65.

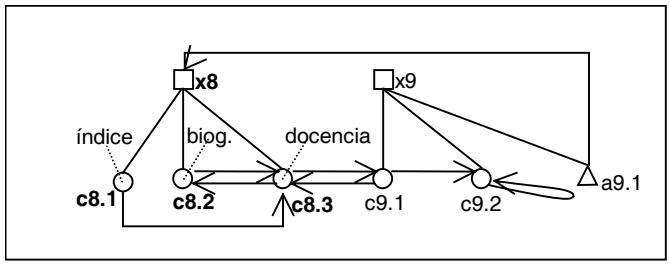

Figura 5.65 Selección de la docencia

- Si ahora seleccionamos el enlace con isg91,

  f((docencia, aIsg91), c8.3, Activos3, Mostrar3)

  $\exists 9.1((c8.3, c9.1, enlace) \in A \wedge enlAct(docencia, aIsg91) \in l(c8.3, c9.1, enlace))$
  $\wedge (c8.3, docencia) \in Mostrar3$

  $\exists 8 \exists 9 ((x8, c8.3, conexión) \in A \wedge (x9, c9.1, conexión) \in A \wedge 8 \neq 9)$

  Activos4 = {x9, c9.1, c9.2}
  Mostrar4 = {(c9.1, isg1), (c9.2, null)}

  Ilustrado en la Figura 5.66.

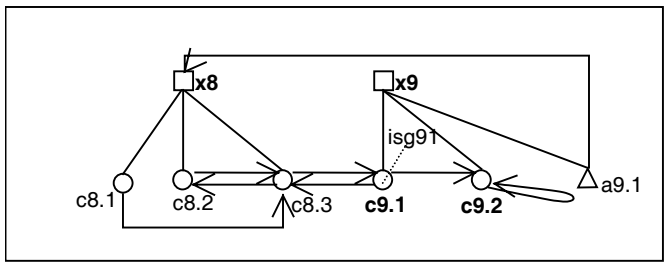

Figura 5.66 Selección de isg91

- Si ahora volvemos la docencia,

  f((isg91, aDoc), c9.1, Activos4, Mostrar4)

  $\exists 8.3((c9.1, c8.3, enlace) \in A \wedge enlAct(isg91, aDoc) \in l(c9.1, c8.3, enlace))$
  $\wedge (c9.1, isg91) \in Mostrar3$

  $\exists 9 \exists 8 ((x9, c9.1, conexión) \in A \wedge (x8, c8.3, conexión) \in A \wedge 9 \neq 8)$

  Activos5 = {x8, c8.1, c8.2, c8.3}

Mostrar5 = {(c8.3, docencia)} ∪ {(c8.1, índice), (c8.2, biografía)}

Ilustrado en la Figura 5.67. No haremos más interacción en esta ventana, pues la parte correspondiente a los contenidos generados dinámicamente que aparecen en la ventana x9 es equivalente a la de la ventana x3.

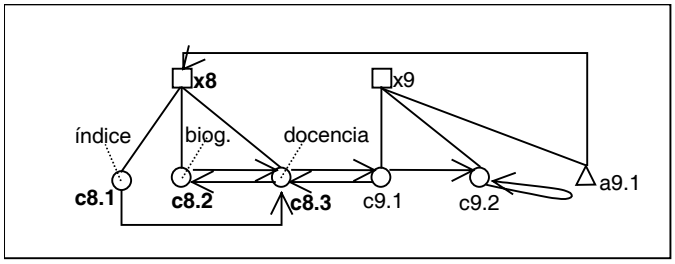

Figura 5.67. Vuelta a docencia

- Finalmente, si hubiésemos optado por la activación del activador de nexo a0.4, tendríamos:

  f(⊥, a0.4, Activos1, Mostrar1) = a(x10), ya que

  ∃10 (a0.4, x10, enlace) ∈ A

  Activos2 = {x10, c10.1, c10.2, c10.3, c10.4, c10.5, c10.6}
  Mostrar2={(c10.1, índice), (c10.2, biografía), (c10.3, null), (c10.4, null), (c10.5, null), (c10.6, null)}

Ilustrado en la Figura 5.68. Además en la Figura 6.23 del Capítulo 6 podemos ver la aplicación correspondiente a la representación Pipe.

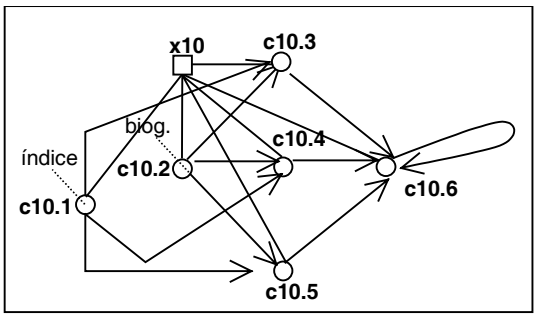

Figura 5.68 Selección última ventana

- Si seleccionamos el enlace de la biografía con la foto de españa.

  f((biografía, aEsp), c10.2, Activos2, Mostrar2)

  ∃10.3((c10.2, c10.3, enlace) ∈ A ∧ enlAct(biografía, aEsp) ∈ l(c10.2, c10.3, enlace))
  ∧ (c10.2, biografía) ∈ Mostrar2

  ∃10 ((x10, c10.2, conexión) ∈ A ∧ (x10, c10.3, conexión) ∈ A)

  Activos3 = Activos2
  Mostrar3 = Mostrar2 \ {(c10.3, null)} ∪ {(c10.3, españa)}

Ilustrado en la figura 5.69.

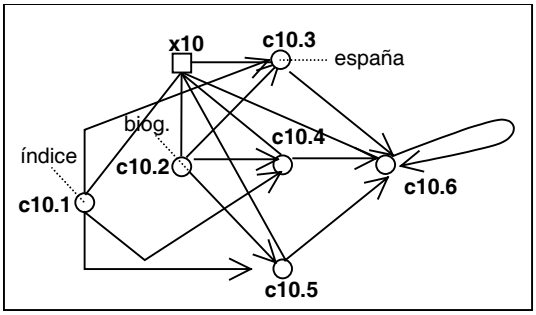

Figura 5.69 Selección de españa

- Si seleccionamos el enlace de españa con madrid.

  f((españa, aMad), c10.3, Activos3, Mostrar3)

  ∃10.6((c10.3, c10.6, enlace) ∈ A ∧ enlAct(españa, aMad) ∈ l(c10.3, c10.6, enlace))
  ∧ (c10.3, españa) ∈ Mostrar3

  ∃10 ((x10, c10.3, conexión) ∈ A ∧ (x10, c10.6, conexión) ∈ A)

  Activos4 = Activos3
  Mostrar4 = Mostrar3 \ {(c10.6, null)} ∪ {(c10.6, madrid)}

Ilustrado en la figura 5.70.

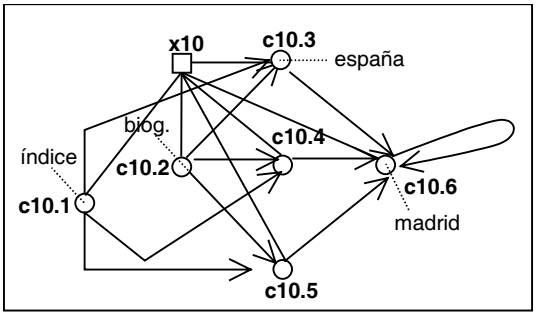

Figura 5.70 Selección de madrid

- Si ahora seleccionamos el enlace del índice con docencia.

  f((índice, aDoc), c10.1, Activos4, Mostrar4)

  ∃10.4((c10.1, c10.4, enlace) ∈ A ∧ enlAct(índice, aDoc) ∈ l(c10.1, c10.4, enlace))
  ∧ (c10.1, índice) ∈ Mostrar4

  ∃10 ((x10, c10.1, conexión) ∈ A ∧ (x10, c10.4, conexión) ∈ A)

  Activos5 = Activos4
  Mostrar5 = Mostrar4 \ {(c10.4, null)} ∪ {(c10.4, docencia)}

Ilustrado en la figura 5.71.

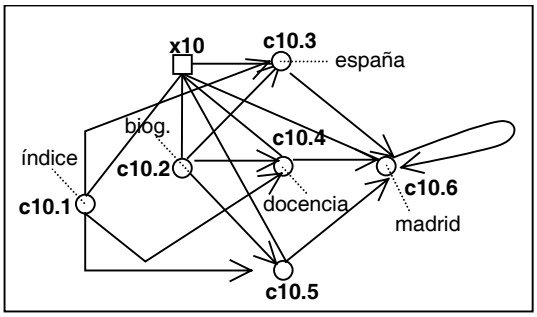

Figura 5.71 Selección de docencia

- Si ahora seleccionamos el enlace de docencia con isg1.

  f((docencia, aIsg1), c10.4, Activos5, Mostrar5)

  $\exists$10.6((c10.4, c10.6, enlace) $\in$ A $\wedge$ enlAct(docencia, aIsg1) $\in$ l(c10.4, c10.6, enlace))
  $\wedge$ (c10.4, docencia) $\in$ Mostrar5

  $\exists$10 ((x10, c10.4, conexión) $\in$ A $\wedge$ (x10, c10.6, conexión) $\in$ A)

  Activos6 = Activos5
  Mostrar6 = Mostrar5 \ {(c10.6, madrid)} $\cup$ {(c10.6, isg91)}

  Ilustrado en la Figura 5.72. No se continúa con la interacción de los contenidos generados dinámicamente al estar en usa situación bastante similar a la de x3.

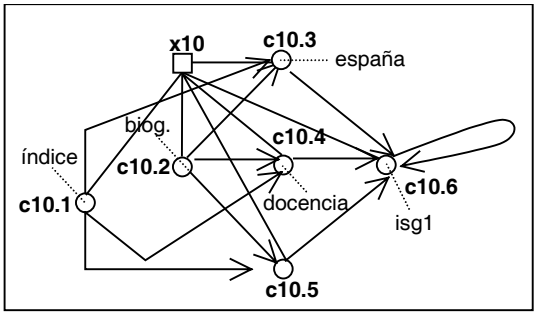

Figura 5.72 Selección de isg1

5.4 Discusión y comparativa

El modelo Pipe surgió como resultado de intentar modelar aplicaciones hipermedia partiendo de las siguientes premisas:

- Separar el nivel de almacenamiento del de presentación, tal y como promueve el modelo Dexter.
- Proporcionar soporte para contexto y sincronización, tal y como promueve el modelo Amsterdam.
- Contar con una semántica de navegación por defecto formalizada, y por tanto contar con la posibilidad de generar prototipos automáticamente.
- Modelar aplicaciones que presenten contenidos estáticos y/o dinámicos.
- Proporcionar soporte a los procesos subordinados.

Compararemos el modelo con los sistemas de representación hipermedia presentados en el Capítulo 2.

Separación de contenidos, navegación y presentación. En el modelo Pipe el nivel de contenidos está caracterizado por los conjuntos de contenidos C_A. El nivel de almacenamiento, está representado por el conjunto de enlaces E_A. A diferencia del modelo Dexter, ahora se permite explícitamente la existencia de contenidos y enlaces generados dinámicamente. El nivel de presentación viene proporcionado por el esquema

y la semántica navegacional. La separación total del esquema navegacional (nodos N y arcos A) de los contenidos hace necesaria la existencia de y un mecanismo de vinculación entre ambos niveles. La función de asignación de contenidos d se encarga de la relacionar nodos con contenidos, mientras que la función de canalización l se encarga de relacionar arcos (tuberías) con enlaces. La información presentacional viene proporcionada por la función p, que se encarga de definir cual va ser la apariencia de los nodos y contenidos. Por último, la semántica de navegación viene proporcionada por las funciones de activación de nexo y de activación de enlace (a y f), además de por los conjuntos Activos y Mostrar. Aunque el modelo no supone ningún tipo de organización sobre los contenidos, en la práctica puede resulta bastante útil imponer cierto tipo de estructura sobre estos. El modelo de proceso PlumbingXJ utilizará XML como mecanismo de estructuración de los mismos. Podíamos decir que desde cierto punto de vista Pipe es un esfuerzo por añadir tiempo, contexto, contenidos dinámicos y semántica de presentación al modelo Dexter (nótese la similitud con el artículo [Hardman94]).

Contexto. El modelo Pipe permite representar sin problemas la noción de contexto. A diferencia de los modelos con representación explícita de eventos (Labyrinth y OOHDM), el contexto no se define recurriendo a formalismos que imposibilitan la existencia de una semántica de navegación por defecto (eventos, y diagramas de contexto + clases de contexto + mapas de ADVs, respectivamente), sino utilizando la estructura de grafo ampliado proporcionada por los conjuntos de nodos y arcos (N y A). A diferencia de los modelos con esquema navegacional definido utilizando formalismos matemáticos más complejos (hipergrafos y Trellis), la noción de contexto queda capturada de una manera mucho más realista, ya que no hay que "forzar los modelos" para que funcionen convenientemente. Finalmente respecto al modelo Amsterdam, Pipe sustituye las jerarquías inducidas por los componentes compuestos por relaciones entre nodos nexos y contenedores.

Sincronización. Las relaciones de sincronización ofrecidas por el modelo Pipe son similares a las que permite el modelo Amsterdam. Ahora al desaparecer el componente compuesto no existe distinción entre la sincronización de grano grueso y grano fino, siendo esta de grano fino. Los arcos de sincronización del modelo Amsterdam están representados por las conexiones sincronizadas y por las funciones que etiquetan dichas conexiones. Sin embargo el modelo Pipe al prescindir de la noción de canal, no garantiza evitar la posible colisión de recursos respecto a un determinado medio (por ejemplo, dos paneles superpuestos).

Semántica de navegación por defecto. El modelo Pipe permite utilizar una estructura de grafo ampliada para representar el esquema navegacional de la aplicación. Nótese que aunque la estructura de grafo pudiera parecer una aproximación similar a la del modelo HAM, el doble nivel de enlaces hace mucho más potente al grafo. Además este doble nivel de enlaces, evita tener que explicitar en el esquema navegacional todos los enlaces a nivel de contenidos, permitiendo una notación más compacta que la ofrecida por el hipergrafo o la red de Petri. En el modelo Pipe la semántica de navegación viene claramente definida a través de los conjuntos Activos y Mostrar, y la evolución de estos en función de los enlaces o conexiones que se activen, en base a las funciones de activación de nodo nexo y de activación de enlace (a y f). Es una semántica sencilla e intuitiva (una vez que obvia el mecanismo matemático subyacente) capaz de representar con elegancia la noción de *frame* (una simplificación del contexto Amsterdam). Utilizando esta semántica de navegación por defecto será posible construir generadores automáticos de prototipos, fundamentales tras la etapa de conceptualización, tal y como demandan [Fraternali 99], [Ginige 97] y [Nanard 98].

Organización relacional de los contenidos. El modelo no presupone ningún tipo de organización de los contenidos. Tampoco hace ninguna suposición sobre la organización interna de estos, más allá de la capacidad de poder asignarles anclas. En cambio permite establecer enlaces entre estos, con independencia del esquema navegacional que se les desee asignar. Nótese que enlaces de contenidos dan lugar a estructuras de grafo, similares a la de la figura 2.10 del Capítulo 2, en vez de relaciones más jerárquicas como las proporcionadas por modelo HDM o RMDM (figuras 2.26 y 2.30 del Capítulo 2, respectivamente).

Además los las relaciones entre contenidos establecidas a través de la función de relación r son más genéricas en Pipe que en modelos hipermedia de corte relacional. En efecto, supongamos que se desea modelar una aplicación hipermedia en la que aparece la biografía de una persona con enlaces a su país de nacimiento, estudios, aficiones, etc. El texto de la biografía no mantiene ninguna relación semántica con los otros contenidos. Por tanto para este tipo de aplicaciones los modelos relacionales no funcionan demasiado bien. Por supuesto se podría haber considerado la relación aparece_citado_en como posible nexo de unión entre la biografía y el resto de contenidos, pero esta relación no es más que la constatación de la existencia de enlaces entre un texto y varios contenidos. Es decir, con el fin de obtener la mayor genericidad posible (tal y como

hace Dexter) en el contexto de las aplicaciones hipermedia pensamos que la relación básica entre los contenidos debe ser la de existencia de enlaces entre contenidos genéricos, y no una relación inducida entre entidades concretas.

Nótese que esta concepción de relación también permite extender la potencia de los modelos relacionales. Si admitimos la relación aparece_citado_en como una relación admisible, esta nos permitiría incluir un único vínculo entre la biografía y el contenido correspondiente. Supongamos que la biografía contiene referencias a una lista de paises visitados por la persona. Dentro de esta lista tenemos los países ordenados por continentes, y en la biografía aparecen enlaces a continentes concretos de esta lista de países. Desde un punto de vista relacional, y en base a que las relaciones se establecen entre entidades (contenidos) solo tendríamos la relación aparece_citado_en entre la biografía y la lista de paises, viéndonos imposibilitados de capturar los enlaces que existen entre la biografía y la lista, y entre la biografía y partes concretas de la lista de paises. Nótese que este es un problema de "anclaje", pero las anclas, como no podía ser de otra manera, son partes bastante espinosas de los modelos relacionales. RMDM opta por utilizar botones que se deducen de la existencia de una relación entre dos entidades (imposibilitando por tanto el ejemplo de la biografía y los paises). HDM opta por una solución similar, teniendo en cuenta, eso si, la mayor complejidad de las entidades en este modelo. Una solución posible sería considerar como campos claves de las entidades los pares formados por el nombre (o ID de la entidad) junto con el ancla correspondiente. Esta aproximación no es la indicada en los modelos RMDM o HDM, y desde el punto de vista relacional no deja de ser un "truco hipermedia", ya que las entidades deberían existir por si solas. Además introduce un gravísimo problema de redundancia. Una solución alternativa sería considerar las anclas como otra entidad (que a su vez se relacionan con las entidades destino) y relacionarlas con las entidades origen de la relación. Esta es una solución que vulnera los principios de diseño del modelo E-R, y que no es propuesta ni en HDM ni en RMDM. Además, en ese diagrama E-R alternativo no aparecería ninguna relación directa entre las entidades semánticamente relacionadas, con la falta de legibilidad que eso conlleva. En el modelo Pipe dicha redundancia no existe, ya que los contenidos C_A se consideran como algo aparte a las relaciones, haciendo estas referencias a los mismos. Es decir, hablando en términos Dexter, en el modelo Pipe el nivel de composición interna es independiente del de almacenamiento, y se relaciona con este a través de las anclas.

Por tanto, en cuanto a la relación de Pipe con los modelos puramente relacionales (RMDM y HDM), este puede verse como una generalización de los anteriores. En efecto, si partimos de una organización relacional de los contenidos podemos optar por canalizar directamente por las tuberías las relaciones entre entidades, que en último término se concretarán en enlaces. De esta forma supongamos que tenemos una empresa formada por departamentos. Estos dirigen proyectos y están formados por empleados, tal y como muestra el diagrama entidad-relación de la Figura 5.73.

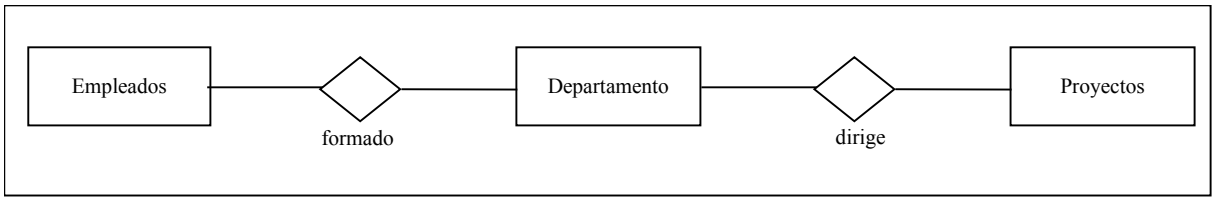

Figura 5.73 Diagrama entidad-relación

Si agrupamos las relaciones en formado, y dirige, podemos encontrarnos con:

dirige = {(ventas, p1), (desarrollo, p2), (desarrollo, p3) }
formado = {(ventas, luis), (desarrollo, ana), (desarrollo, pepe), (personal, marta)}

Supuesto que las anteriores relaciones, realmente codifican al conjunto de enlaces (obviando como se asignan realmente las anclas a las entidades, como por ejemplo las anclas aP1, y aLuis, ancladas sobre ventas), las anteriores relaciones representan el conjunto de enlaces (supuesto unidireccionalidad en la relación):

E_A = { (ventas, aP1, p1, total), (desarrollo, aP2, p2, total), (desarrollo, aP3, p3, total), (ventas, aLuis, luis, total), (desarrollo, aAna, ana, total), (desarrollo, aPepe, pepe, total), (personal, aMarta, marta, total) }

mostrado en la Figura 5.70.

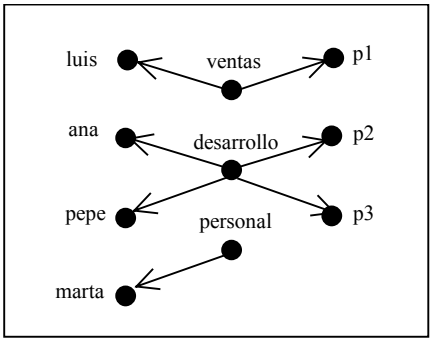

Figura 5.74 Grafo de contenido inducido por las relaciones

Si queremos indicar que tendremos una ventana formada por cuatro paneles una para listar los departamentos de la empresa (es decir, algo así como un nodo índice del modelo RMDM), otra para los departamentos, otra para personal, y otra para proyectos, no tenemos más que considerar el esquema navegacional descrito por los conjuntos N, y A (la Figura 5.75 recoge este esquema):

N = {x1, c1, c2, c3, c4}
A= { (x1, c1, conexión), (x1, c2, conexión), (x1, c3, conexión), (x1, c4, conexión),
(c1, c2, enlace), (c2, c3, enlace), (c2, c4, enlace) }

y que ampliar el conjunto de contenidos con índice, y el conjunto de enlaces con

i = {(índice, aVent, ventas, total), (índice, aDes, desarrollo, total), (índice, aPers, personal, total) }

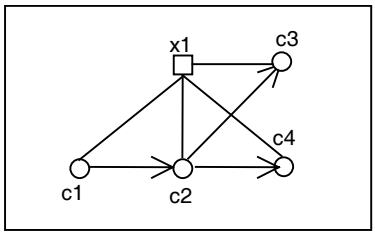

Figura 5.75 Esquema navegacional

Finalmente no tenemos más que definir las funciones d, y l. En efecto:

d = {(**c1**, índice, ∅), (**c2**, null, {ventas, desarrollo, personal), (**c3**, null, {p1, p2, p3}),
(**c4**, null, {ana, luis, marta, pepe}) }

l = {(**c1, c2, enlace**, i), (**c2, c3, enlace**, dirige), (**c2, c4, enlace**, formado)}

Es decir, utilizamos las relaciones, que pueden entenderse como subconjuntos del conjunto E_A para facilitar la canalización de los enlaces entre contenidos. Esta aproximación puede emplearse en las técnicas Pipe siempre que se desee. En la formulación hemos optado por no agrupar los enlaces por relaciones para obtener una mayor genericidad del modelo. Podríamos decir para simplificar que en los modelos relacionales puros las relaciones inducen los enlaces, mientras que en Pipe son los enlaces los que (en caso que nos interese) inducen relaciones.

Gestión explícita de eventos. Pipe no se preocupa de modelar los eventos que sucedan en el sistema más allá de la activación de enlaces. Este hecho se debe a que un modelo que pretenda proporcionar una semántica de navegación por defecto no puede proporcionar soporte explícito para modelar dichos eventos (a otro nivel de abstracción, una situación similar es la de SMIL frente a XUL). Esto no excluye la aparición dentro de Pipe de procesos subordinados, es decir, pequeñas aplicaciones que se ejecutan dentro de una aplicación hipermedia. Lo que sucede es que estos procesos en ningún caso pueden actuar sobre el esquema navegacional de la aplicación. Por ejemplo, Pipe es capaz de modelar una página HTML con un programa en JavaScript que se encarga de ejecutar un ejercicio de un alumno. Lo que no es capaz de modelar, es el caso

anterior, donde el ejercicio es capaz de activar por si mismos enlaces y llevarnos a otra ventana distinta de la aplicación. En estos casos sería necesario utilizar modelos, más potentes en este aspecto, como Labyrinth u OOHDM. El precio a pagar será una pérdida de la semántica de navegación por defecto.

Gestión de usuarios y seguridad. En principio Pipe no tiene en cuenta ni la gestión de usuarios ni la seguridad. Gracias a su formalización no sería muy complejo extensiones formales para la gestión de usuarios y seguridad similares a las proporcionadas por el modelo Labyrinth.

Distinción de enlaces a nivel contenidos y esquema navegacional. Este modelo es junto con el modelo OOHDM el único que permite tener un doble nivel de enlaces, a nivel contenidos (similar al propuesto por el modelo HAM) y a nivel esquema navegacional (impuesto sobre el anterior). Esto es una importante ventaja, ya que permite expresar las conexiones semánticas existentes entre los contenidos con independencia del esquema navegacional (representado por los conjuntos de nodos y arcos N y A) que se les quiera asignar a estos (el ejemplo 5.3 ilustra este hecho). Ahora la asignación de contenidos y enlaces a nodos y enlaces navegacionales quedan recogidas por las funciones asignación de contenidos (d) y la función de canalización de enlaces (l), respectivamente. Además la semántica de navegación inducida a partir del esquema navegacional queda recogida por los conjuntos Activos y Mostrar, junto a las funciones de activación de nodos y enlaces, frente a las clases de contexto y los mapas ADVs del modelo OOHDM. Por último, como hemos comentado ya, esta doble jerarquía de enlaces permite representar de forma simple y elegante la noción de contexto, evitando el tener que mostrar todas las relaciones semánticas entre contenidos a nivel navegacional.

Técnicas orientadas a objetos. Como ya hemos comentado, el modelo no se encarga de representar explícitamente ninguna actividad computacional ajena a la sustancialmente hipermedia. Por tanto no ve necesario ningún formalismo para representar actividades computacionales genéricas. Sin embargo, a nivel práctico, y debido a la inclusión de procesos subordinados en las aplicaciones hipermedia, se hace necesario el utilizar algún tipo de representación. Por ejemplo, en la fase de diseño del modelo de proceso PlumbingXJ es posible utilizar diagramas UML para modelar a los procesos subordinados.

Contenidos generados dinámicamente. Pipe es el único modelo con semántica de navegación por defecto capaz de caracterizar contenidos generados dinámicamente. OOHDM y Labyrinth pueden utilizarse para conseguir esta representación, pero debido a su genericidad no son capaces de capturar explícitamente cual es la semántica de presentación de dichos contenidos. Por supuesto, haciendo una interpretación "amplia" del modelo Dexter, puede entenderse que este también es capaz de modelar contenidos generados dinámicamente.

La Figura 5.76 retoma la Figura 2.58 del Capítulo 2 para hacer explícitas las relaciones entre Pipe y las características principales de los modelos considerados en esta tesis.
Finalmente cabe remarcar que al igual que modelos como RMDM, Labyrinth u OOHDM, el modelo Pipe ha sido creado como guía de un modelo de proceso específico para la construcción de aplicaciones hipermedia, proporcionando la abstracción y semántica suficientes para guiar la construcción de tales aplicaciones. Este será el contenido de los siguientes capítulos.

	a	b	c	d	e	f	g	h	i	j
Dexter	✓	×	×	×	×	×	×✓	×	×	×
Amsterdam	✓	✓	✓	×✓	×	×	×	×	×	×
HAM	×	×	×	✓	×	×	×	×✓	×	×
Hipergrafos	✓	×✓	×	✓	×	×	×	×✓	×	×
Trellis	✓	×✓	×✓	✓	×	×	×	×✓	×	×
HDM	✓	×	×	×✓	✓	×	×	×	×	×
RMM	✓	×	×	×✓	✓	×	×✓	×	×✓	×
Labyrinth	✓	✓	✓	×	×	✓	✓	✓	×	×✓
OOHDM	✓	✓	✓	×	×✓	✓	✓	×	✓	✓
Pipe	✓	✓	✓	✓	×	×	✓	×	✓	×

a) Separación contenidos, navegación y presentación
b) Contexto
c) Sincronización
d) Formalización semántica de navegación
e) Organización relacional de los contenidos
f) Gestión explícita de eventos
g) Contenidos dinámicos
h) Gestión de usuarios y seguridad
i) Distinción enlaces entre contenidos y navegacionales
j) Técnicas orientadas a objetos

✓: Si
×: No
×✓: Parcialmente o con ligeras modificaciones

Figura 5.76 Comparación de Pipe con el resto de sistemas de representación

6 Los modelos de proceso Plumbing y PlumbingXJ

6.1 Introducción

En este capítulo presentaremos el modelo de proceso *Plumbing* [Navarro 02], resultante de utilizar el modelo hipermedia Pipe como guía para la fase de conceptualización del modelo de proceso mixto de Fraternali y Ginige-Lowe (Capítulo 3). Lo seguimos denominando modelo de proceso y no metodología porque no se compromete ninguna técnica de implementación concreta, sino que solamente se propone un mecanismo concreto para cubrir la fase de conceptualización. A nuestro entender esto no compromete la naturaleza abstracta del modelo de proceso, al igual que la inclusión de técnicas concretas de especificación no compromete la naturaleza abstracta de modelos de proceso genéricos [Pressman 01], [Sommerville 01]. Nótese que incluso el uso de una notación concreta de modelado, y de unas técnicas de programación específicas no comprometen la naturaleza de modelo de proceso del Modelo de Proceso Unificado [Jacobson 00].

Además concretaremos el modelo de proceso Plumbing en *PlumbingXJ* [Navarro 02], modelo que utiliza las capacidades proporcionadas por XML y Java para facilitar la etapa de prototipado. Por las misma razones argumentada en el párrafo anterior, seguimos considerándolo modelo de proceso, y no metodología.

Pipe, Plumbing, y PlumbingXJ representan la última evolución de una línea de investigación que comenzó en 1998 con el trabajo de tercer ciclo *Aplicaciones de los lenguajes de marcado en la abstracción del diseño de un sistema hipermedia* [Navarro 98]. En este trabajo se estudiaba la aplicabilidad de los lenguajes de marcado para la construcción de aplicaciones hipermedia, basándose en los desarrollos previos de [Fernández 97], [Fernández 98b], y [Fernández 98d]. En dicho trabajo de tercer ciclo ya se insinuaba la posibilidad de codificar un sistema de representación hipermedia mediante documentos XML. También se utilizaba la técnica de *sobremarcado* aunque en una versión primitiva (en este capítulo veremos dicha técnica). La evolución lógica de los conceptos allí presentados fue concretada en una metodología de desarrollo hipermedia, aplicada fundamentalmente al entorno educativo [Navarro 00a], [Navarro 00b], [Navarro 00c]. Aumentando el nivel de abstracción de la aproximación anterior se decidió considerar además de una metodología de desarrollo, un modelo de proceso que guiase de forma genérica la construcción de aplicaciones hipermedia [Navarro 00d], [Navarro 01a], [Navarro 01b], [Navarro 01c]. En esta etapa también se propuso una evolución de la técnica de sobremarcado hasta su estado actual. Como nos dimos cuenta que el modelo de proceso propuesto era bastante similar al presente en la literatura hipermedia, finalmente optamos por utilizar uno ya existente, adaptándolo con el modelo hipermedia y las técnicas de construcción de desarrollo propio. También optamos por restringir la aplicación de nuestra aproximación a las fases de conceptualización y prototipado exclusivamente [Navarro 02]. Precisamente, esta tesis recoge los últimos resultados a este respecto.

6.2 El modelo de proceso Plumbing

En el Capítulo 3 vimos el modelo de proceso resultante de incorporar ciertas características del modelo de Ginige-Lowe al modelo de Fraternalli. Dicho modelo de proceso era básicamente un modelo de construcción de prototipos, con una fase previa de conceptualización y prototipado cuyo objetivo es facilitar las etapas de diseño y construcción en detalle. La Figura 6.1 ilustra este modelo de proceso.

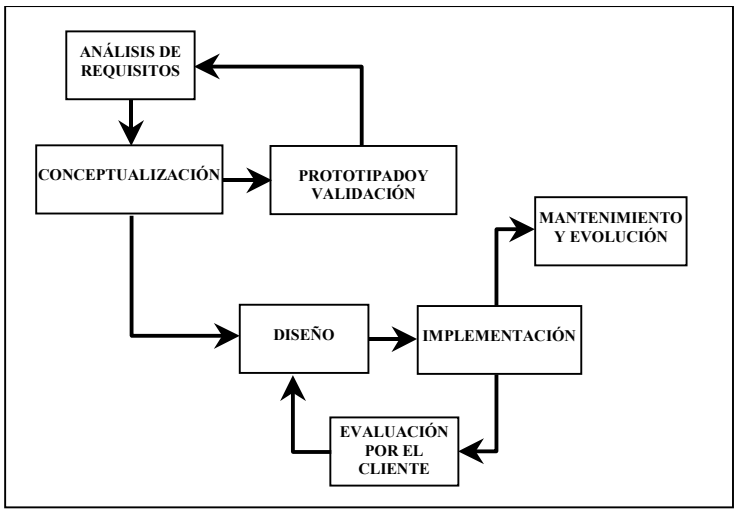

Figura 6.1 Fusión del modelo de proceso Fraternalli con el de Ginige y Lowe

En el Capítulo 5 definíamos el modelo hipermedia Pipe, creado específicamente para cubrir la fase de conceptualización del modelo de proceso anterior. Incorporando dicho sistema de representación hipermedia en el modelo de proceso de Fraternali obtenemos el modelo de proceso Plumbing, descrito en la Figura 6.2.

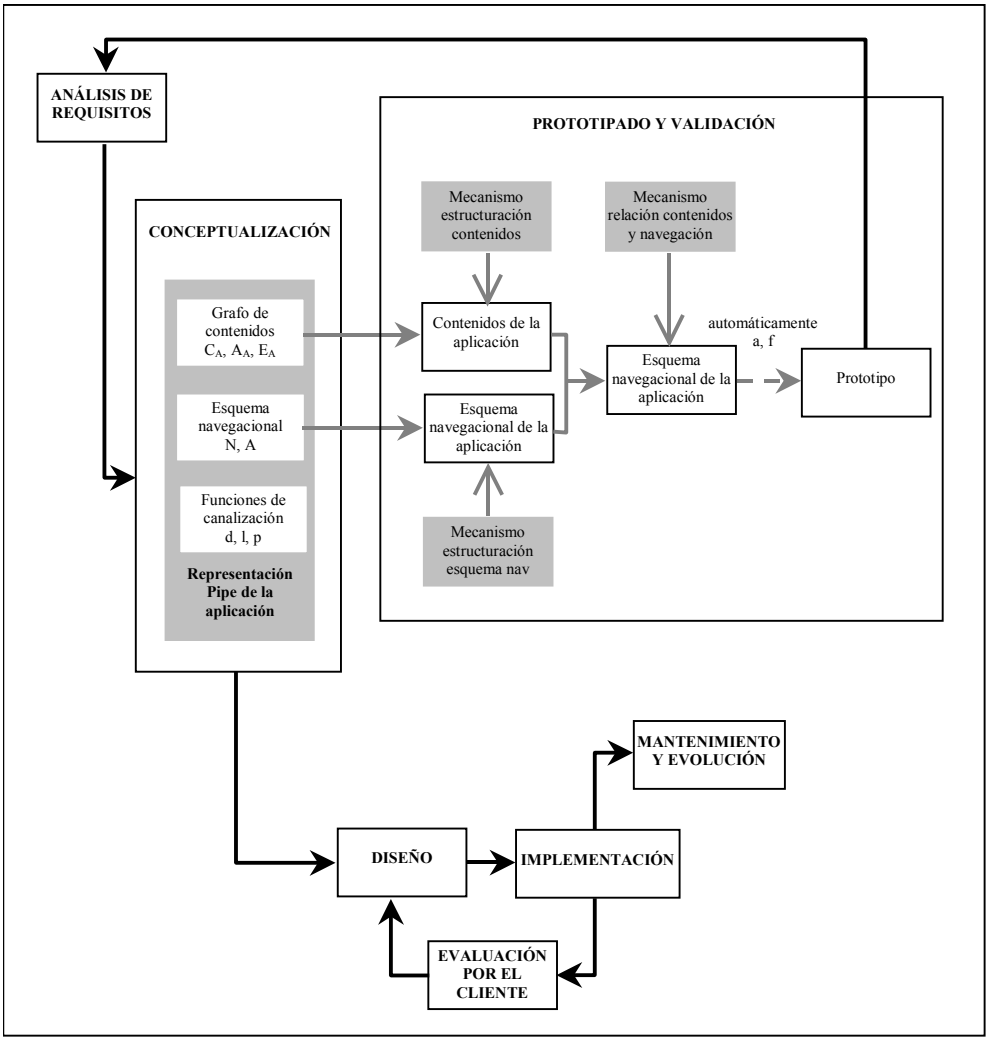

Figura 6.2 Modelo de proceso Plumbing

Análisis de requisitos y conceptualización

Al igual que en el modelo de Fraternali, tras la fase de análisis de requisitos viene la fase de conceptualización. En esta etapa los desarrolladores deben proporcionar una representación de la aplicación basada en las estructuras Pipe. Es importante darse cuenta de la trascendencia de esta fase en el contexto hipermedia, y de como el modelo Pipe puede ayudar a su exitosa consecución. En las aplicaciones software tradicionales la fase previa de análisis de requisitos produce como norma general el *documento de especificación de requisitos* [Somerville 01], en el cual constan las funcionalidades que el sistema debe llevar a cabo, incluyendo restricciones y rendimiento. En el caso de las aplicaciones hipermedia el documento de especificación de requisitos tiene una naturaleza especial. En efecto la funcionalidad de toda aplicación hipermedia siempre es la misma: van a existir contenidos enlazados sobre los cuales se va a imponer un esquema navegacional que va a caracterizar su acceso. Aunque la frase anterior es correcta tiene bastante poco valor como documento de especificación de requisitos. Por tanto ¿qué información debería incluir dicho documento para ser de utilidad en el proceso de desarrollo del software? Cualquier persona que se haya visto involucrada en el desarrollo de aplicaciones hipermedia sabe que cual es la naturaleza primordial de la información proporcionada en las especificación de requisitos (en el apartado D.2 del Apéndice D puede verse un caso real de guión de dicha especificación). Esta, la mayoría de las veces, es del tipo: "quiero una ventana que aparezca dividida en dos paneles. En el primer panel va a aparecer el índice. Los contenidos seleccionados desde el índice aparecerán en el segundo panel. A su vez cuando seleccione estos contenidos el destino del enlace aparecerá en otra ventana dividida en tres paneles, de tal forma que... " Si la aplicación es de un tamaño medio, tras cinco ventanas y dieciséis paneles, el documento de especificación de requisitos parecerá cualquier cosa menos una especificación legible de la aplicación hipermedia.

Es por esta razón que se hace necesario un mecanismo auxiliar que permita expresar toda esta información hipermedia, independiente en principio de características de diseño tales como procesamiento distribuido, desarrollo basado en agentes, o plataforma de implementación. Precisamente, esta es la razón de ser de Pipe: capturar la especificación estrictamente hipermedia de manera precisa, y de tal forma que no se comprometa ninguna característica de implementación. Así, en la etapa de *conceptualización* se deberá proceder a identificar el grafo de contenidos de la aplicación hipermedia, es decir, los conjuntos C_A, A_A, y E_A. Como Pipe está basado en los preceptos del modelo Dexter, establecer el conjunto de contenidos C_A es equivalente a proporcionar el nivel de composición interna, aunque a un nivel muy abstracto. El conjunto de enlaces E_A representa al nivel de almacenamiento, siendo el conjunto de anclas A_A la manera de relacionar ambos niveles.

Una vez establecidos los contenidos y sus relaciones sin ningún genero de dudas se procederá a especificar el esquema navegacional del que va a estar formado la aplicación, es decir, los conjuntos N y A. Precisamente cuando establezcamos el conjunto de nodos N y de arcos A entre los mismos estaremos sentando las bases del nivel de ejecución Dexter. Llegados a este punto, la relación entre el esquema navegacional y el grafo de contenidos que se establece a través de las funciones de canalización, es decir las funciones de asignación de contenidos, de canalización y opcionalmente de presentación (d, l, y p, respectivamente), puede especificarse al mismo tiempo que el esquema navegacional, o en un paso independiente. En aras de una mayor generalidad, la Figura 6.2 lo muestra como actividades independientes.

Por lo tanto al finalizar la etapa de conceptualización se debería contar con una representación Pipe completa de la aplicación hipermedia. Es importante remarcar que a pesar del carácter formal del modelo hipermedia somos conscientes que una especificación manual de los conjuntos que se ven involucrados en una descripción Pipe (C_A, A_A, E_A, N, A, d, l, y p) puede ser, y de hecho es, una labor tediosa. Con el fin de aliviar este problema estamos desarrollando la herramienta CASE PlumbingMatic, la cual será capaz de construir de manera visual tanto el grafo de contenidos como el esquema navegacional. Por supuesto dicha herramienta también debe ser capaz de relacionar ambos niveles mediante las funciones de canalización. Este es uno de los puntos de trabajo futuro primordiales de esta tesis.

Prototipado y validación

Aunque en principio la representación Pipe de la aplicación pudiera servir como paso previo al diseño hay que tener en cuenta una constante en el desarrollo de software: el cambio en los requisitos por parte del cliente [Pressman 01]. En el caso de las aplicaciones hipermedia este cambio puede afectar a: (i) los contenidos; (ii) los enlaces establecidos entre estos; (iii) el esquema navegacional impuesto sobre el grafo de contenidos. Además estos cambios pueden producirse de manera asilada o de forma simultánea. En estas

condiciones comenzar el diseño y construcción en detalle de una aplicación hipermedia puede ser una tarea costosa que puede llevarnos al fracaso del proyecto de software [Fraternali 99].
Precisamente, la fase de *prototipado y validación* tiene como objetivo el proporcionar una "especificación de requisitos" estable que permita abordar el diseño y desarrollo con mayores garantías de éxito. Evidentemente, esta fase debe ser capaz de reutilizar la representación Pipe a la hora de generar los prototipos. En el caso de la creación de los contenidos la información proporcionada por el modelo hipermedia será bastante escasa, ya que precisamente Pipe obvia tal caracterización. Lo que sí sería deseable es contar con un mecanismo de estructuración de los contenidos que sea lo suficientemente potente como para especificar tanto los contenidos como los enlaces establecidos entre estos. Lenguajes como XML o estructuras relacionales pueden ser ejemplos claros de estos formalismos. Evidentemente, y aunque en el modelo de proceso de desarrollo aun nos encontremos bastante lejos de la etapa de implementación, sería deseable utilizar un formalismo que estimemos pueda ser reutilizado directamente en el diseño y construcción de la aplicación (como el relacional), o lo suficientemente genérico y potente como para ser un formato de representación de la información que pueda traducirse sin demasiados esfuerzos a la representación final de los datos (como XML).

En esta etapa también debemos ser capaces de proporcionar un formalismo que sea capaz de representar el esquema navegacional, así como los mecanismos de relación entre el nivel de contenidos y el de navegación. Al igual que en la etapa de conceptualización podemos especificar en un mismo paso el esquema navegacional y su relación con el grafo de contenidos, u optar por mecanismos de representación independientes. Formalismos adecuados para representar el esquema navegacional y su relación con el grafo de contenidos pueden ser primitivas de acceso del estilo RMDM o lenguajes ad hoc que permitan especificar la información que necesitamos codificar.

Hemos hecho hincapié en la existencia de mecanismos formales bien definidos en la fase de prototipado para caracterizar las descripciones Pipe por dos razones. La primera es la ventaja de disponer de un mecanismo formal de especificación que permita mostrar sin ningún género de duda cual es la información que se está codificando. La segunda es poder disponer de los ingredientes necesarios para la generación automática de prototipos basadas en las codificaciones de las representaciones Pipe.

El ciclo formado por las etapas de análisis de requisitos, conceptualización y prototipado se itera hasta que se dispone de una especificación de requisitos de la aplicación lo suficientemente estable como para iniciar el diseño detallado y construcción de la misma. A diferencia de las aplicaciones tradicionales, y a pesar de haber sido más costoso el proceso de especificación de requisitos, ahora contamos con una aplicación razonablemente estable, y de la cual disponemos de gran parte de sus contenidos. Respecto a los mismos, y en lo referente a la parte de los procesos subordinados, en la etapa de prototipado y validación puede optarse por utilizar representaciones UML o incluso código ejecutable (acompañado por supuestos de diagramas UML) para caracterizarlos. La decisión final depende del nivel de profundidad al que se quiera llegar. Nótese que al ser los procesos subordinados aplicaciones "estancas" que no afectan a la navegación, su especificación, diseño y desarrollo no es una tarea muy costosa.

En cuanto a las aplicaciones hipermedia que presenten contenidos dinámicos puede optarse por una simulación basada en contenidos estáticos, o en la creación del código que implemente dicho comportamiento dinámico. En este caso deben establecerse mecanismos de integración en la parte de generación de prototipos para conseguir un automatismo los suficientemente avanzado. En este sentido el trabajo de [Sierra 01] puede ser una técnica natural de integración.

En cuanto al resto de fases, Plumbing no introduce ninguna modificación a las descritas en el modelo de Fraternali, pudiendo producirse eso sí, acoplamientos intencionados entre las fases de conceptualización y prototipado con diseño e implementación para facilitar el desarrollo del software. Esta es una decisión que cada organización de desarrollo de software debe considerar.

6.3 El modelo de proceso PlumbingXJ

6.3.1 PlumbingXJ

El modelo de proceso Plumbing no comprometía ninguna técnica específica para proporcionar representaciones concretas de las estructuras Pipe generadas en la fase de conceptualización. Tampoco especificaba como podían generarse prototipos de la aplicación de manera automática a partir de dicha representación de las estructuras Pipe. *PlumbingXJ* es una especialización del modelo de proceso Plumbing en el cual se utiliza XML para representar la información proporcionada en la fase de conceptualización. Además ahora se dispone de un *Generador Automático de Protipos* Java (*GAP*) para construir los prototipos a partir de las descripciones XML. La Figura 6.3 recoge la aproximación.

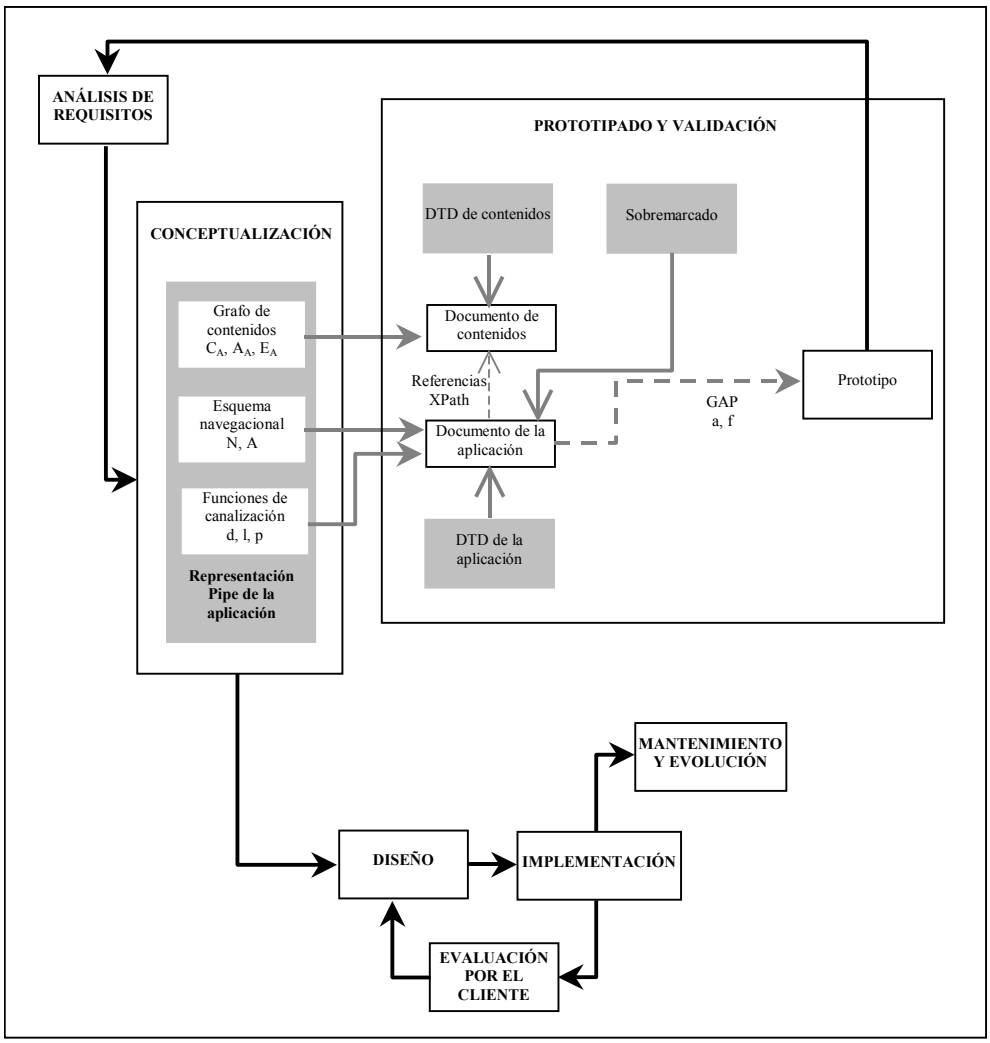

Figura 6.3 Modelo de proceso PlumbingXJ

Tal y como podemos comprobar en dicha figura, los únicos cambios entre Plumbing y PlumbingXJ se producen en la fase de prototipado y validación. En Plumbing decíamos que los formalismos utilizados para proporcionar representaciones concretas de las estructuras Pipe debían estar ligados a las técnicas de implementación finales, o ser lo suficientemente genéricos como para permitir una rápida transición a sistemas de representación concretos. En particular nosotros hemos optado por la segunda opción en aras de una mayor aplicabilidad de PlumbingXJ, eligiendo a XML como mecanismo de representación de Pipe por dos razones. La primera es que en la actualidad XML es un formato de intercambio universal que facilita la comunicación entre sistemas [Bryan 98], [Peat 97]. La segunda radica en el alto poder de estructuración de este tipo de lenguajes [Sperberg-McQueen 94], el cual los hace idóneos para caracterizar aplicaciones hipermedia [Navarro 01a].

DTD de contenidos

Por lo tanto el mecanismo para estructurar el grafo de contenidos de la aplicación hipermedia va a ser una DTD XML a la cual denominaremos *dtd de contenidos*. Esta DTD es distinta para cada aplicación hipermedia y permite imponer una doble estructura a los contenidos de la aplicación. La primera es la del grafo de contenidos. La segunda es la proporcionada por las relaciones jerárquicas inducidas a partir de las relaciones existentes entre los elementos XML. En Pipe, los contenidos eran independientes de las anclas impuestas sobre estos y de los enlaces originados a partir de estas anclas. Aunque esta es la aproximación que en principio parece más correcta, y que sería implementable en mecanismos de marcado a través del estándar XLink, por ahora hemos preferido no seguir esta solución. En su lugar hemos optado por incluir la información de anclaje y enlaces en la propia estructura de contenidos mediante atributos `ID/IDREF`. La razón fundamental de este hecho radica en la falta de soporte CASE, la cual se hace casi indispensable para especificar cuales son las anclas y los enlaces de contenidos con total independencia respecto a estos. Cuando la herramienta CASE PlumbingMatic esté disponible optaremos por la opción XLink.

A la instancia de la DTD de contenidos la denominaremos *documento de contenidos*. Dicho documento proporciona una representación XML del grafo de contenidos Pipe (es decir, los conjuntos C_A, A_A, y E_A). Veremos más adelante en este capítulo que tipo de contenidos se pueden incluir en esta instancia y como se especifican anclas y enlaces entre ellos. En lo referente a los procesos subordinados, y tal y como propone Plumbing, caben dos opciones. La primera es proporcionar su código en una clase Java, el cual es invocado dinámicamente desde el generador automático de prototipos (veremos esto más adelante en este capítulo). La segunda es proporcionar un diagrama UML e incluirlo como imagen en la aplicación. Todo depende del nivel de compromiso en el desarrollo de este tipo de aplicaciones que se quiera alcanzar en la fase de prototipado.

En lo referente a los contenidos generados dinámicamente por ahora la única solución que proporciona PlumbingXJ es su simulación a través de contenidos XML (veremos un ejemplo más adelante). Como trabajo futuro queda la inclusión de un mecanismo de generación dinámica de contenidos y enlaces integrable directamente con el generador automático de prototipos. Aunque todavía no hemos comenzado el análisis de dicha solución, el mecanismo de integración de procesos subordinados, parece una opción razonable de implementación.

DTD de la aplicación. Sobremarcado

Al igual que en el caso de los contenidos, el mecanismo de estructuración del esquema navegacional va a ser una DTD XML a la cual denominaremos *DTD de la aplicación*. A diferencia de la DTD de contenidos, la DTD de la aplicación es única, ya que en principio la estructura del esquema navegacional viene totalmente determinada por Pipe. Tal y como veremos la terminología utilizada por esta DTD está más cercana al lenguaje de descripción de interfaces (es decir, ventana, panel, etc.) que a la terminología Pipe (es decir, nodo nexo, nodo contenedor, etc.).

Aunque en el modelo de proceso Plumbing la especificación del esquema navegacional y su relación con el grafo de contenidos eran procesos independientes, en PlumbingXJ se concibe como un único proceso (compárense las figuras 6.2 y 6.3). En efecto, en PlumbingXJ cuando se crea la instancia de la DTD de la aplicación para describir el esquema navegacional, no solo se está describiendo la estructura de interfaz gráfica de usuario junto con los caminos navegacionales sino que además se proporciona la información de canalización del grafo de contenidos mediante el esquema navegacional. El mecanismo utilizado para conseguir esta relación tiene el nombre de *sobremarcado*.

La técnica de sobremarcado es bastante sencilla. Cuando se procede a la descripción del esquema navegacional mediante la instancia de la DTD de la aplicación, el contenido real de los elementos que representan a los nodos contenedores son referencias XPath a elementos del documento de contenidos. De esta forma codificamos la función de asignación de contenidos Pipe. La función de canalización de enlaces viene representada por elementos de la DTD de la aplicación que crean las tuberías por las que se canalizarán los enlaces de contenidos. Veremos en profundidad estas cuestiones a lo largo del capítulo. Si fuera necesario adaptar los contenidos para su inclusión en el esquema navegacional, una opción factible sería la sustitución de expresiones XPath por expresiones XSLT. Otra opción válida hubiese sido la inclusión de expresiones

XML Query [W3C XMLQ], pero hemos optado por XPath debido razones de estandarización a la hora de escribir esta tesis.

En lo referente a la función de presentación, por ahora no la hemos prestado excesiva atención, ya que solamente estamos centrados en la etapa de prototipado. En cualquier caso, la inclusión de hojas de estilo CSS junto con transformaciones XSLT específicas podría ser una solución válida.

A la instancia de la DTD de la aplicación la denominaremos *documento de la aplicación* y será la entrada para el Generador Automático de Prototipos encargado de producir el prototipo de la aplicación hipermedia a través de la representación XML de las estructuras Pipe. Dicho documento de la aplicación codifica al esquema navegacional Pipe, y a las funciones de canalización (es decir, a los conjuntos y funciones N, A, d, l). Por ahora la generación de este documento de la aplicación es manual, pero cuando el soporte CASE PlumbingMatic esté disponible esta generación será automática. De esta forma se utilizará la DTD de la aplicación de una manera similar a la DTD SMIL, cuyos sistemas de autoría permiten generar su instancia bien de forma manual en modo texto, o bien de manera gráfica basada las estructuras del modelo hipermedia en el que está basada (Amsterdam, en este caso particular).

Generador Automático de Prototipos, GAP

GAP es una aplicación Java que produce el prototipo de la aplicación hipermedia a partir del documento de la aplicación. Por tanto, GAP no es más que un *sistema hipermedia* [Halasz 94] en el que la descripción de la aplicación hipermedia se produce mediante un documento XML.

La implementación de GAP está basada en el API Swing de Java y en los APIs DOM y XSLT de Alphaworks. Hemos decidido obviar en este capítulo una descripción en detalle del sistema GAP, centrándonos exclusivamente en los modelos de proceso y en la DTD de la aplicación. En el Apéndice B puede encontrarse el diseño y código del Generador Automático de Prototipos.

6.3.2 La DTD de la aplicación

Como ya hemos comentado en el apartado anterior, la DTD de la aplicación es la encargada de proporcionar estructura al esquema navegacional de la aplicación y a las funciones de canalización del modelo Pipe. Aunque Pipe utiliza una terminología independiente de cualquier interfaz de usuario, en PlumbingXJ hemos optado por concretar una terminología específica que facilite la aplicación por personas externas al área computacional [Navarro 01a]. Tal y como mostrábamos en la Tabla 5.4 del Capítulo 5, a los nodos nexos los denominaremos *ventanas*, a los nodos contenedores los denominaremos *paneles*, y a los nodos activadores de nexos los denominaremos *activadores de ventanas*. La Figura 6.4 ilustra la DTD de la aplicación PlumbingXJ.

```
<!ELEMENT aplicacion (nombre?, ventana+)>
<!ATTLIST aplicacion id ID #REQUIRED>

<!ELEMENT nombre (#PCDATA)>
<!ELEMENT ventana (nombre?, panel*, activadorV*)>
<!ATTLIST ventana id ID #REQUIRED>

<!ELEMENT panel (contenidoPanel, destinoEnlaces?)>
<!ATTLIST panel id ID #REQUIRED
                 tam CDATA "300, 200">

<!ELEMENT contenidoPanel (contenidoDefecto?, contenido*, contenidoGrupo*)>
<!ELEMENT contenidoDefecto (#PCDATA)>
<!ATTLIST contenidoDefecto tipo (XPath|proceso) "XPath">
<!ELEMENT contenido (#PCDATA)>
<!ATTLIST contenido tipo (XPath|proceso) "XPath">
<!ELEMENT contenidoGrupo (#PCDATA)>

<!ELEMENT destinoEnlaces EMPTY>
<!ATTLIST destinoEnlaces panel IDREF #REQUIRED
>

<!ELEMENT activadorV (nombre)>
<!ATTLIST activadorV id ID #REQUIRED
                      ventana IDREF #REQUIRED>
```

Figura 6.4 DTD de la aplicación PlumbingXJ

Si hubiésemos optado por una representación lo más parecida posible a las estructuras Pipe habríamos obtenido una DTD como la mostrada en la Figura 6.5.

```
<!ELEMENT Pipe (C, E, N, A, d, l, p)>
<!ELEMENT C (contenido+)>
<!ELEMENT contenido (#PCDATA)>
<!-- Referenciamos el archivo con los contenidos -->
<!ATTLIST contenido id ID #REQUIRED>
<!-- Identificamos los contenidos en la instancia -->
<!ELEMENT E (enlace+)>
<!ELEMENT enlace (anclaO, anclaD)>
<!ATTLIST enlace id IDREF #REQUIRED
                 origen IDREF #REQUIRED
                 destino IDREF #REQUIRED>
<!ELEMENT anclaO (#PCDATA)>
<!ELEMENT anclaD (#PCDATA)>
<!-- El enlace referencia los extremos e incluye anclas -->
<!ELEMENT N (nodo+)>
<!ELEMENT nodo EMPTY>
<!ATTLIST nodo id ID #REQUIRED
               tipo (contenedor|nexo) #REQUIRED>
<!-- Los nodos solo tienen un identificador -->
<!ELEMENT A (arco+)>
<!ELEMENT arco EMPTY>
<!ATTLIST arco id ID #REQUIRED
              origen IDREF #REQUIRED
              destino IDREF #REQUIRED
              tipo (conexion|enlace|conexionS|enlaceS) conexion>
<!-- Los arcos enlazan nodos y añaden tipo -->
<!ELEMENT d (imagenD+)>
<!ELEMENT imagenD EMPTY>
<!ATTLIST imagenD nodo IDREF
                  contenido IDREFS>
<!-- A cada nodo se le asigna el conjunto de contenidos -->
<!ELEMENT l (imagenL+)>
<!ELEMENT imagenL EMPTY>
<!ATTLIST imagenL arco IDREF
                  enlaces IDREFS>
<!--A cada enlace se le asigna el conjunto de enlaces de contenido -->
<!ELEMENT p (imagenP+)>
<!ELEMENT imagenP (#PCDATA)>
<!ATTLIST imagenP nodo IDREF>
<!-- A cada nodo se le asigna su presentación (en lenguaje natural) -->
```

Figura 6.5 Posible DTD de la aplicación utilizando terminología Pipe

Optamos por la primera opción, ya que a nuestro entender facilita la aplicación de PlumbingXJ. A continuación estudiaremos cada uno de los elementos la DTD de la aplicación (mostrada en la Figura 6.4).

aplicacion

```
<!ELEMENT aplicacion (nombre?, ventana+)>
<!ATTLIST aplicacion id ID #REQUIRED>
```

Este es el elemento raíz del documento de la aplicación. Está formado por el nombre de la aplicación opcional (`nombre`) y una sucesión de ventanas (`ventana+`). Es precisamente esta enumeración de ventanas la que permite representar las estructuras del modelo Pipe, ya que tal y como hemos comentado, una ventana representa a un nodo contenedor.

nombre

```
<!ELEMENT nombre (#PCDATA)>
```

Describe el nombre de la aplicación, así como los títulos de las ventanas y de los activadores de botones.

ventana

```
<!ELEMENT ventana (nombre?, panel*, activadorV*)>
<!ATTLIST ventana id ID #REQUIRED>
```

Este elemento representa las ventanas de la aplicación, o en términos Pipe los nodos nexos de la misma. Está formado por un nombre (`nombre`), una sucesión de paneles (`panel*`), y una sucesión de activadores de ventana (`activadorV*`). Aunque la DTD lo permite no tiene sentido considerar una ventana sin paneles ni activadores. Se ha decidido no restringir esta posibilidad desde la DTD para facilitar la legibilidad de la misma.

Ahora las relaciones jerárquicas existentes entre una ventana y los elementos que la componen permiten especificar las relaciones estructurales básicas del modelo Pipe, o dicho de otra forma las conexiones entre un nodo nexo (ventana), y los nodos contenedores (paneles) y activadores de nexos (activadores de ventanas). Por ahora la DTD de la aplicación solo implementa las relaciones estructurales básicas, sin dar soporte a las conexiones temporales entre un nodo nexo y el resto de nodos. Este aspecto queda como trabajo futuro de la tesis.

Por razones de implementación de la versión actual de GAP las ventanas deben tener asignado un identificador (`id`) de la forma v_i. Precisamente este identificador permitirá referenciarlas desde los nodos activadores de ventana. Aunque no es necesario seguir un orden específico a la hora de numerarlas, si es importante no dejar numeraciones incompletas (por ejemplo v_1, v_3, v_4, en vez de v_1, v_2, v_3). Además, la ventana inicial debe ser la v_1.

panel

```
<!ELEMENT panel (contenidoPanel, destinoEnlaces?)>
<!ATTLIST panel id ID #REQUIRED
                tam CDATA "300, 200">
```

Este elemento representa un panel interior a una ventana, o en términos Pipe a un nodo contenedor. Tal y como comentamos en el apartado anterior, el documento de la aplicación no solo proporciona una representación al esquema navegacional sino que al mismo tiempo codifica las funciones de canalización. Precisamente el elemento `contenidoPanel` permite especificar la expresión XPath que implementa la técnica de sobremarcado.

El elemento `destinoEnlaces` es el encargado de establecer una tubería entre este nodo contenedor y el nodo contenedor destino. A diferencia del modelo Pipe que permitía especificar varias tuberías con origen en el mismo nodo contenedor, por ahora, en PlumbingXJ un nodo contenedor solamente puede ser origen de una tubería (aunque si puede ser destino de varias). De esta forma el canalizado de los enlaces de contenidos se hace de manera automática, ya que todos los enlaces de contenidos con origen en un contenido asignado a un nodo contenedor serán canalizados por la única tubería de la que puede ser origen. De no ser así, se debería proveer un mecanismo de enumeración de enlaces de contenidos para indicar por que tubería son canalizados. A este respecto, y a falta de soporte CASE, la agrupación de enlaces de contenidos tal y como se comenta en el apartado 5.4 del Capítulo 5 podría ser de bastante utilidad.

Los paneles están identificados unívocamente (`id`), para permitir ser referenciados como destino de una tubería. Al igual que las ventanas siguen una numeración, siguiendo en este caso el formato $p_{i.j}$, donde la `i` representa la ventana que lo contiene, y la `j` el número de panel. Al igual que con las ventanas no hay un orden especial para numerar los paneles de una ventana, pero no se puede saltar la numeración de los mismos. También tiene un atributo para determinar su tamaño (`tam`). Respecto a su posición, por ahora GAP distribuye los paneles de manera horizontal. Próximas implementaciones permitirán especificar la posición en coordenadas absolutas.

contenidoPanel

```
<!ELEMENT contenidoPanel (contenidoDefecto?, contenido*, contenidoGrupo*)>
```

Este elemento representa al contenido de un panel, o en términos Pipe codifica a la función de asignación de contenidos d. En particular un panel puede tener un contenido por defecto opcional (`contenidoDefecto`), una lista de los contenidos que van a aparecer en el panel (`contenido*`), y una lista de elementos para procesar contenidos en grupo (`contenidoGrupo*`). Como veremos más adelante, este elemento no restringe los contenidos que pueden aparecer en un panel, sino que proporciona a GAP una manera de procesar los contenidos de la aplicación.

Nótese que aunque la DTD permite no especificar ningún contenido para el elemento `contenidoPanel`, esto no tiene demasiado sentido desde el punto de vista práctico. Se ha decidido no restringir esta posibilidad en la DTD para facilitar la legibilidad de la misma.

contenidoDefecto

```
<!ELEMENT contenidoDefecto (#PCDATA)>
<!ATTLIST contenidoDefecto tipo (XPath|proceso) "XPath">
```

Representa el contenido por defecto del panel, o en términos Pipe el contenido por defecto del nodo contenedor. Por tanto este es el contenido que mostrará el panel cuando sea activado. Si no se especifica este elemento muestra un panel en blanco (el contenido null de Pipe). También incluye un atributo (`tipo`) que indica si el contenido va a ser una expresión XPath o un proceso subordinado. En el primer caso el contenido del elemento es una referencia XPath al documento de contenidos. Esta referencia se utiliza para seleccionar los contenidos del documento de contenidos que deben aparecer en el panel. Dicho elemento de contenidos debe tener un atributo de nombre `id`, de tipo `ID`. En el segundo caso contiene el nombre de la clase que representa al proceso subordinado. En este caso no se induce ningún tratamiento GAP. Para un correcto funcionamiento en la invocación del proceso subordinado, el mismo debe ser una implementación del interfaz `ProcesoSubordinado` siguiente.

```
public interface ProcesoSubordinado {
    public void fijarPanel(JPanel p);
    public void activar ();
    public void parar();
}
```

Donde la función `fijarPanel()` proporciona un panel para que el proceso subordinado construya su interfaz de usuario, `activar()` construye la interfaz gráfica de la clase sobre el panel indicado por el documento de la aplicación, y la función `parar()` detiene cualquier actividad computacional que esté llevando a cabo el proceso. El Apéndice B proporciona más detalles de funcionamiento. Aunque podíamos haber optado por comprometer más la implementación de los procesos subordinados utilizando una clase abstracta Java en lugar de un interfaz hemos optado por la segunda opción, con el fin de limitar lo menos posible la implementación de los procesos subordinados.

Ya estamos en condiciones de proporcionar una ventana de la aplicación Pipe. En particular va a ser una ventana formada por dos paneles. Cuando se active la ventana en el primer panel (`p1.1`) se mostrará una foto de Carmen Electra, y en el segundo (`p1.2`) un proceso subordinado que permite sumar dos números. En efecto la siguiente instancia de la DTD de la aplicación proporciona esta ventana.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="figura6_6">

   <ventana id= "v1">
      <nombre>Ventana1</nombre>

     <panel id="p1.1" tam="155, 155">
       <contenidoPanel>
        <contenidoDefecto>/contenidos/foto</contenidoDefecto>
       </contenidoPanel>
     </panel>
```

```
      <panel id= "p1.2">
        <contenidoPanel>
         <contenidoDefect
        </contenidoPanel>
      </panel>
   </ventana>

</aplicacion>
```

Para el siguiente conjunto de c

```
<?xml version="1.0" encod

<!DOCTYPE contenidos [
       <!ELEMENT contenic
       <!ELEMENT foto EM
       <!ATTLIST foto ima
]>

<contenidos>
  <foto imagen="c:\plumbi
</contenidos>
```

Nótese que no se ha incluido ...iento de contenidos, ya que ningún contenido enlaza c

Nótese también que en el docu... la interfaz del proceso subordinado. Esto se debe a ...faz. Tal y como hemos comentado, otra opción viable ...una imagen, en vez del propio código que da lugar al p...

contenido

```
<!ELEMENT contenido (#PCDATA)>
<!ATTLIST contenido tipo (XPath|proceso) "XPath">
```

Este elemento se encarga de asignar los contenidos (salvo el contenido por defecto) a un panel (o nodo nexo). Al igual que en `contenidoDefecto` puede ser una expresión XPath que induce un tratamiento en el documento de contenidos (como antes el contenido debe tener un elemento `id` de tipo `ID`), o el nombre de una clase que implementa a `ProcesoSubordinado`. En este caso (y a diferencia del elemento `contenidoDefecto`) la inclusión del nombre de dicha clase tiene un valor meramente declarativo, ya que podría obviarse obteniendo los mismos resultados. En el Apéndice B se explica el porque de esta situación.

Cuando se desee que un contenido por defecto también pueda ser referenciado desde otro contenido, es necesario incluir dicho contenido tanto en la etiqueta `contenidoDefecto` como en la etiqueta contenido. Las razones detalladas de este hecho pueden encontrarse en el Apéndice B. Un ejemplo de esto puede encontrarse en el ejemplo del elemento `destinoEnlaces`.

Nótese que en GAP no hace falta el cierre transitivo de la relación de enlaces para asignar contenidos a un nodo. Esto se debe a que en GAP la asignación de contenidos induce la generación de páginas HTML en vez de restringir los contenidos que aparecen dentro de un nodo. Por tanto no es necesario enumerar los contenidos que van a aparecer en un panel, si estos ya han sido generados. Para más detalles consultar el Apéndice B.

contenidoGrupo

```
<!ELEMENT contenidoGrupo (#PCDATA)>
```

Este elemento facilita el tratamiento de contenidos de la forma `<!ELEMENT items (item+)>`. De esta forma cuando se especifica:

```
<contenidoGrupo>/contenidos/items</contenidoGrupo>
```

se obtiene los mismos resultados que si se especificase:

```
<contenido>/contenidos/item[1]</contenido>
<contenido>/contenidos/item[2]</contenido>
                .....................
<contenido>/contenidos/item[n]</contenido>
```

supuesto que hubiese `n` `items`. Al igual que con contenido, los `item` deben tener un atributo `id` de tipo `ID`.

destinoEnlaces

```
<!ELEMENT destinoEnlaces EMPTY>
<!ATTLIST destinoEnlaces panel IDREF #REQUIRED>
```

Este elemento se encarga de especificar el panel destino de los enlaces con origen en el panel que lo contiene, o en términos Pipe, la función de canalización I. Como ya hemos comentado, y a diferencia de Pipe, cada panel solo puede ser origen de una tubería (aunque si puede ser destino de varias). De esta forma la canalización de los enlaces de contenidos se realiza de manera automática por la única tubería disponible. De no ser así habría que asignar explícitamente los enlaces de contenidos a tuberías, cuestión esta que dejamos en suspenso hasta la disponibilidad de PlumbingMatic.

Al igual que con los contenidos ahora no es necesario calcular el cierre transitivo de la relación de enlaces. Esto se debe a que la canalización en GAP no restringe los enlaces que pueden pasar por una tubería, simplemente indica el panel destino del enlace. Para más detalles consultar el Apéndice B.

Supongamos ahora que tenemos una aplicación formada por dos textos. En el primero se referencia al segundo, y viceversa. Además queremos que los textos aparezcan en paneles de dos ventanas distintas. En este caso no tenemos más que considerar el siguiente documento de la aplicación.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="figura6_7">

   <ventana id= "v1">
      <nombre>ventana 1</nombre>
      <panel id= "p1.1">
        <contenidoPanel>
         <contenidoDefecto>/contenidos/texto1</contenidoDefecto>
          <contenido>/contenidos/texto1</contenido>
        </contenidoPanel>
        <destinoEnlaces panel="p2.1"/>
      </panel>
   </ventana>

   <ventana id= "v2">
```

```
      <nombre>ventana 2</nombre>
      <panel id= "p2.1">
        <contenidoPanel>
         <contenido>/contenidos/texto2</contenido>
        </contenidoPanel>
        <destinoEnlaces panel="p1.1"/>
      </panel>
   </ventana>

</aplicacion>
```

Que opera sobre el siguiente documento de contenidos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE contenidos [
       <!ELEMENT contenidos (texto1, texto2)>
       <!ELEMENT texto1 (#PCDATA|referencia)*>
       <!ATTLIST texto1 id ID #REQUIRED>
       <!ELEMENT texto2 (#PCDATA|referencia)*>
       <!ATTLIST texto2 id ID #REQUIRED>
       <!ELEMENT referencia (#PCDATA)>
       <!ATTLIST referencia href IDREF #REQUIRED>
]>

<contenidos>
       <texto1 id="contenido1">En este texto aparece una referencia al <referencia
href="contenido2">segundo texto</referencia></texto1>
       <texto2 id="contenido2">En este texto aparece una referencia al <referencia
href="contenido1">primer texto</referencia></texto2>
</contenidos>
```

Las ventanas generadas pueden verse en la Figura 6.7.

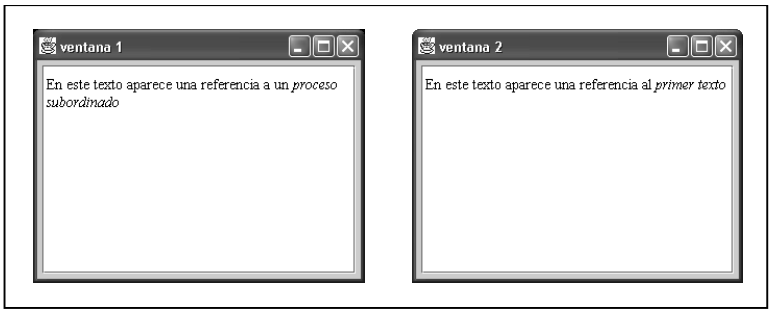

Figura 6.7 Ventanas generadas. La segunda aparece al seleccionar el enlace de contenidos en el primer texto (en cursiva), tras lo cual desaparece la primera. Esta vuelve a aparecer tras seleccionar el enlace de contenidos del segundo texto (en cursiva), tras lo cual desaparece la segunda.

Tal y como comentamos en PlumbingXJ hemos optado por incluir directamente las anclas (caracterizadas mediante el atributo `href`) en los contenidos. En el apartado 6.3.3 pueden encontrarse las restricciones de uso, y en el Apéndice B el porque de estas restricciones.

activadorV

```
<!ELEMENT activadorV (nombre)>
<!ATTLIST activadorV id ID #REQUIRED
                     ventana IDREF #REQUIRED>
```

Este elemento se encarga de incluir un activador de ventana, o en términos Pipe un activador de nodo nexo. Solamente hay que especificar el nombre del activador (`nombre`) y la ventana a activar (`ventana`). Aunque en este caso no es necesario utilizar ninguna nomenclatura especial para identificarlos (`id`) por motivos de consistencia utilizaremos la misma que para identificar paneles, es decir, $p_{i.j}$.

Así, si tenemos un texto que enlaza con un proceso subordinado y queremos mostrar ambos contenidos en dos ventanas distintas, con acceso a la primera desde la segunda mediante un activador de ventana, no tenemos más que considerar el siguiente documento de la aplicación.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="figura6_8">

   <ventana id= "v1">
      <nombre>ventana 1</nombre>
      <panel id= "p1.1">
        <contenidoPanel>
         <contenidoDefecto>/contenidos/texto</contenidoDefecto>
        </contenidoPanel>
      <destinoEnlaces panel="p2.1"/>

      </panel>
   </ventana>


   <ventana id= "v2">
      <nombre>ventana 2</nombre>
      <panel id= "p2.1">
        <contenidoPanel>
         <contenido tipo="proceso">PShola</contenido>
        </contenidoPanel>

      </panel>
        <activadorV id="av2.1" ventana="v1">
          <nombre>A ventana 1</nombre>
        </activadorV>
   </ventana>

</aplicacion>
```

Que opera sobre el siguiente documento de contenidos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE contenidos [
       <!ELEMENT contenidos (texto)>
       <!ELEMENT texto (#PCDATA|referencia)*>
       <!ELEMENT referencia (#PCDATA)>
       <!ATTLIST referencia hrefP CDATA #REQUIRED>
]>

<contenidos>
       <texto>En este texto aparece una referencia a un <referencia
hrefP="PShola">proceso subordinado</referencia></texto>
</contenidos>
```

En la Figura 6.8 pueden apreciarse las ventanas generadas.

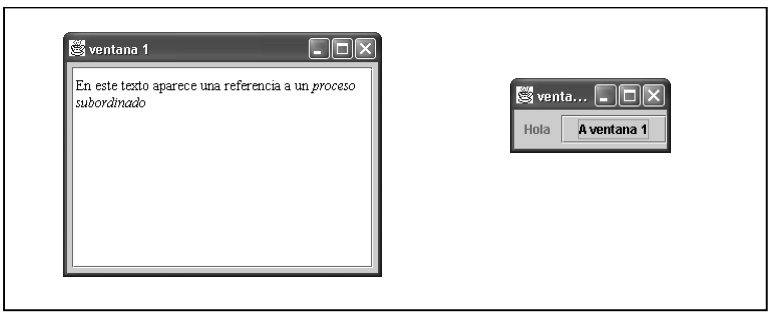

Figura 6.8 Ventanas generadas. La segunda aparece al seleccionar el enlace de contenidos en el texto, tras lo cual desaparece la primera. Esta vuelve a aparecer tras seleccionar el activador de ventana de la segunda, tras lo cual desaparece esta.

6.3.3 Restricciones de uso

Existen una serie de restricciones, o normas de uso para la creación del documento de contenidos y de la aplicación debidas a la implementación del generador de prototipos GAP proporcionada. Estas son:

- Restricciones en el documento de la aplicación:
  - Las ventanas deben numerarse de forma consecutiva y con la nomenclatura v_i.
  - Los paneles deben numerarse de forma consecutiva con la nomenclatura $p_{i.j}$ donde la `i` denota la ventana a la que pertenece y la `j` el número de panel.
  - En los activadores de ventana no es necesario utilizar una nomenclatura especial, pero por consistencia se utilizara $av_{i.j}$.
  - La ventana inicial es la v_1.
  - Si se tiene un único panel para mostrar los contenidos, y se va a referenciar el contenido por defecto del panel desde otro contenido es necesario incluir al contenido por defecto del panel como `contenido`. Esto se debe a que cuando se genera un contenido por defecto a la página HTML generada se le asigna el nombre del panel al cual se le asigna, pero si es referenciada desde los contenidos, entonces se referencia a una página HTML cuyo nombre es su identificador. Para generar esta página es necesario asignarle como `contenido`.

- Restricciones en el documento de contenidos:
  - Los contenidos referenciados mediante etiquetas `contenido` deben tener un atributo `id` de tipo `ID`. Lo mismo le debe suceder a los subelementos referenciados desde una etiqueta `contenidoGrupo`.
  - Los espacios en blanco de los elementos referenciado desde cualquier etiqueta de `contenidoPanel` son tenidos en cuenta.
  - En los contenidos, los atributos de tipo `ID` que sean destinos de enlaces deben llamarse `id`, y solamente puede utilizarse este nombre para este tipo de atributos. De esta forma GAP puede encontrarlos y utilizarlos como nombre de las páginas HTML generadas, con el fin de que sean referenciadas.
  - Las anclas origen deben tener como nombre de atributo origen `href` de tipo `IDREF`.
  - Las anclas destino deben tener como nombre de atributo destino `name` de tipo `CDATA`. Para referenciarlas debe utilizarse la nomenclatura `href="idEleto#ancla"`, donde `idEleto` es el valor del atributo `id` de tipo `ID` del elemento que contiene al elemento con el atributo `name`.
  - Las imágenes deben ser referenciadas a través de un atributo `imagen` de tipo `CDATA`.
  - Para incluir anclas en las imágenes se debe seguir un esquema:

```
<nombreEleto1 imagen="foto.jpg">
      <nombreEleto2 coords="c1" href="r1"/>
       ..............
      <nombreEleto2 coords="cn" href="cn"/>
</nombreEleto1>
```

    para obtener:

```
<img src="foto.jpg" usemap="#foto.jpg"/>
 <map>
   <area shape="poly" coords="c1" href="r1"/>
    .......
   <area shape="poly" coords="cn" href="rn"/>
 </map>
```

  - Las anclas origen que tengan como destino un proceso subordinado deben tener como nombre de atributo origen `hrefP` de tipo `CDATA`.

Como describe el Apéndice B, todas estas restricciones en el documento de contenidos sirven para que funcione correctamente la transformación genérica XSLT que genera los contenidos a partir del documento de contenidos.

6.3.4 Potencia expresiva

La potencia expresiva de GAP es inferior a la proporcionada por Pipe. En particular GAP no es capaz de:

- Representar contenidos dinámicos
- Representar enlaces n-arios.
- Utilizar dos tuberías distintas con origen en el mismo nodo.
- Ignorar enlaces de contenidos.
- Soportar las capacidades de sincronización de Pipe.
- Incluir todos los tipos de contenidos considerados por Pipe.

Veamos cuales pueden sor posibles soluciones, a tener en cuenta como trabajo futuro. Respecto a la primera limitación la línea de investigación debería ser similar a la propuesta para la inclusión de procesos subordinados (ver Apéndice B). La segunda es un problema ligado a las tecnologías subyacentes en Pipe. Una solución posible es la de generar un enlace cuyo destino fuese una página auxiliar que contuviese todos los enlaces los cuales deberían ser seleccionados por el usuario. La tercera limitación obliga a identificar que enlace está canalizado por cada tubería que parte del mismo nodo. Por lo tanto se necesitaría poder asignarlo enlace por enlace, y guardar dicha lista para gestionar la activación. La cuarta pasaría por una enumeración explícita de los enlaces de contenidos que se quieren obviar. La quinta obliga a una gestión del tiempo por parte de GAP para actualizar la aplicación. Finalmente, la sexta se resuelve incluyendo la etiqueta correspondiente y su transformación a su etiqueta funcional HTML.

Todas estas limitaciones se consideran como trabajo futuro de esta tesis.

6.3.5 Ejemplo

A continuación veremos la representación a través del modelo, del ejemplo que vimos en el capítulo sobre el modelo Pipe en la sección 5.3. Debido a la menor potencia expresiva nos hemos visto obligados a hacer una serie de modificaciones respecto al ejemplo allí propuesto. Veamos a continuación los resultados.

Para modelar los contenidos de la aplicación hemos optado por utilizar la siguiente DTD de contenidos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ENTITY % textoA "(#PCDATA|refA)*">
<!ELEMENT refA (#PCDATA)>
<!ATTLIST refA href CDATA #REQUIRED>

<!ELEMENT profesor (índice, biografía, españa, estudios, docencia, formulario,
localidades, asignaturas, resultado)>

<!ELEMENT índice (entrada+)>
<!ELEMENT entrada (#PCDATA)>
<!ATTLIST entrada href CDATA #REQUIRED>

<!ELEMENT biografía %textoA;>
<!ATTLIST biografía id ID #REQUIRED>

<!ELEMENT españa (ancla+)>
<!ATTLIST españa imagen CDATA #REQUIRED
                 id ID #REQUIRED>
<!ELEMENT ancla EMPTY>
<!ATTLIST ancla coords CDATA #REQUIRED
                  href IDREF #REQUIRED>

<!ELEMENT estudios (#PCDATA)>
<!ATTLIST estudios id ID #REQUIRED>

<!ELEMENT docencia (#PCDATA|refA|desD)*>
<!ATTLIST docencia id ID #REQUIRED>
<!ELEMENT desD (#PCDATA)>
<!ATTLIST desD name CDATA #REQUIRED>

<!ELEMENT formulario %textoA;>
```

```
<!ATTLIST formulario id ID #REQUIRED>


<!ELEMENT localidades (localidad+)>
<!ELEMENT localidad (#PCDATA)>
<!ATTLIST localidad nombre CDATA #REQUIRED
                        id ID #REQUIRED>

<!ELEMENT asignaturas (asignatura+)>
<!ELEMENT asignatura (#PCDATA)>
<!ATTLIST asignatura nombre CDATA #REQUIRED
                         id ID #REQUIRED>

<!ELEMENT resultado %textoA;>
<!ATTLIST resultado id ID #REQUIRED>
```

La instancia de dicha DTD puede encontrarse en el Apéndice C (hemos intentado simplificar al máximo los contenidos para facilitar su legibilidad). Debido a la imposibilidad de Pipe para representar tuberías distintas que parten del mismo contenido hemos simplificado el grafo de contenidos original al mostrado por la Figura 6.11. En dicha simplificación hemos eliminado los enlaces bidirreccionales entre los contenidos. Nótese que el problema no radica en bidireccionalidad de los mismos, sino en no poder utilizar dos tuberías distintas con origen en el mismo nodo nexo. De hecho la aplicación generada en la Figura 6.9 tiene enlaces de contenidos bidireccionales.

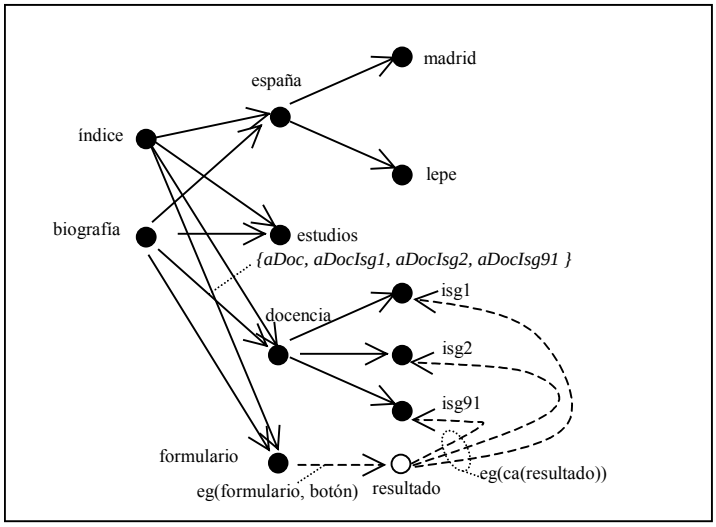

Figura 6.9 Grafo de contenidos

Por otro lado, la parte dinámica de la aplicación ha sido "simulada" por los siguientes contenidos.

```
<formulario id="formulario">Se supone que aquí aparece un formulario de búsqueda, con un
campo de entrada y un <refA href="resultado">botón</refA>.
  </formulario>

.....................

  <resultado id="resultado">Se supone que este es el resultado de la búsqueda que enlaza
con los contenidos. Por ejemplo
con <refA href="isg1">Ingeniería del Software de Gestión I</refA>, <refA
href="isg2">Ingeniería del Software de Gestión II</refA>,
y con <refA href="isg91">Ingeniería del Software de Gestión 91</refA>
  </resultado>
```

Veamos ahora una por una las interfaces de la aplicación en su versión Pipe, documento de la aplicación y GAP. A la ventana original del ejemplo, le vamos a eliminar los dos contenidos de "decorado" y vamos a eliminar su conexión sincro. La Figura 6.10 ilustra los cambios.

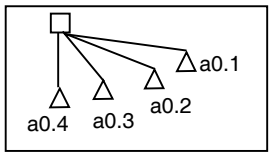

Figura 6.10 Ventana inicial

La parte del documento de la aplicación que representa a esta ventana es la siguiente.

```
<ventana id="v1">
  <nombre>Ventana 0</nombre>
    <activadorV id="av1.1" ventana="v2">
     <nombre>Interfaz 1</nombre>
  </activadorV>
  <activadorV id="av1.2" ventana="v3">
    <nombre>Interfaz 2</nombre>
  </activadorV>
  <activadorV id="av1.3" ventana="v5">
    <nombre>Interfaz 3</nombre>
  </activadorV>
  <activadorV id="av1.4" ventana="v9">
    <nombre>Interfaz 4</nombre>
  </activadorV>
  <activadorV id="av1.5" ventana="v11">
    <nombre>Interfaz 5</nombre>
  </activadorV>
</ventana>
```

Nótese que hay un pequeño desajuste numérico, ya que los nodos $p_{i.j}$ en Pipe se corresponden con las ventanas $v_{i+1.j}$ GAP. La Figura 6.11 muestra la ventana GAP generada.

Figura 6.11 Ventana generada

El primer interfaz que considerábamos en el ejemplo se corresponde con la Figura 6.12.

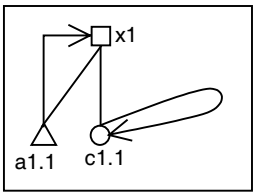

Figura 6.12 Interfaz 1

Su parte del documento de la aplicación es la siguiente.

```
<ventana id= "v2">
      <nombre>Ventana 1</nombre>
      <panel id= "p2.1">
        <contenidoPanel>
          <contenidoDefecto>/profesor/biografía</contenidoDefecto>
          <contenido>/profesor/biografía</contenido>
           <contenido>/profesor/españa</contenido>
           <contenido>/profesor/estudios</contenido>
           <contenido>/profesor/docencia</contenido>
           <contenido>/profesor/formulario</contenido>
           <contenido>/profesor/resultado</contenido>
```

```
            <contenidoGrupo>/profesor/localidades</contenidoGrupo>
            <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
          </contenidoPanel>
          <destinoEnlaces panel="p2.1"/>
      </panel>
      <activadorV id="av2.1" ventana="v2">
         <nombre>A biografía</nombre>
      </activadorV>
</ventana>
```

La ventana generada puede verse en la Figura 6.13.

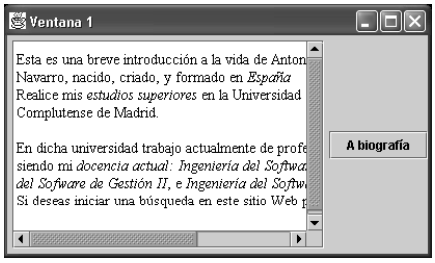

Figura 6.13 Ventana generada

La Figura 6.14 representa la segunda interfaz del ejemplo.

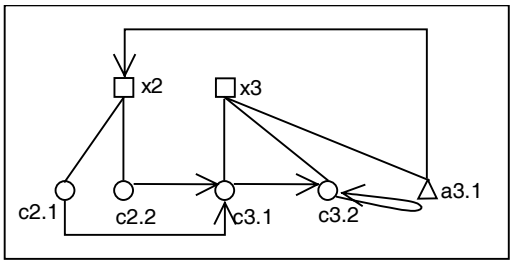

Figura 6.14 Interfaz 2

Su parte del documento de la aplicación es el siguiente.

```
<ventana id= "v3">
      <nombre>Ventana 2</nombre>

      <panel id= "p3.1" tam= "155, 265">
        <contenidoPanel>
          <contenidoDefecto>/profesor/índice</contenidoDefecto>
        </contenidoPanel>
        <destinoEnlaces panel="p4.1"/>
      </panel>

      <panel id="p2.2" tam="400, 265">
        <contenidoPanel>
            <contenidoDefecto>/profesor/biografía</contenidoDefecto>
            <contenido>/profesor/biografía</contenido>
        </contenidoPanel>
        <destinoEnlaces panel="p4.1"/>
       </panel>
   </ventana>

   <ventana id="v4">
     <nombre>Ventana 3</nombre>
     <panel id="p4.1" tam="445, 380">
        <contenidoPanel>
          <contenido>/profesor/españa</contenido>
           <contenido>/profesor/estudios</contenido>
           <contenido>/profesor/docencia</contenido>
```

```
          <contenido>/profesor/formulario</contenido>
          <contenidoGrupo>/profesor/localidades</contenidoGrupo>
          <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
        </contenidoPanel>
       <destinoEnlaces panel="p4.2"/>
    </panel>
    <panel id="p4.2" tam="265, 150">
      <contenidoPanel>
          <contenido>/profesor/resultado</contenido>
      </contenidoPanel>
      <destinoEnlaces panel="p4.2"/>
    </panel>

     <activadorV id="av4.1" ventana="v3">
      <nombre>A ventana 2</nombre>
    </activadorV>
  </ventana>
```

Las ventanas generadas durante la interacción pueden verse en la Figura 6.15.

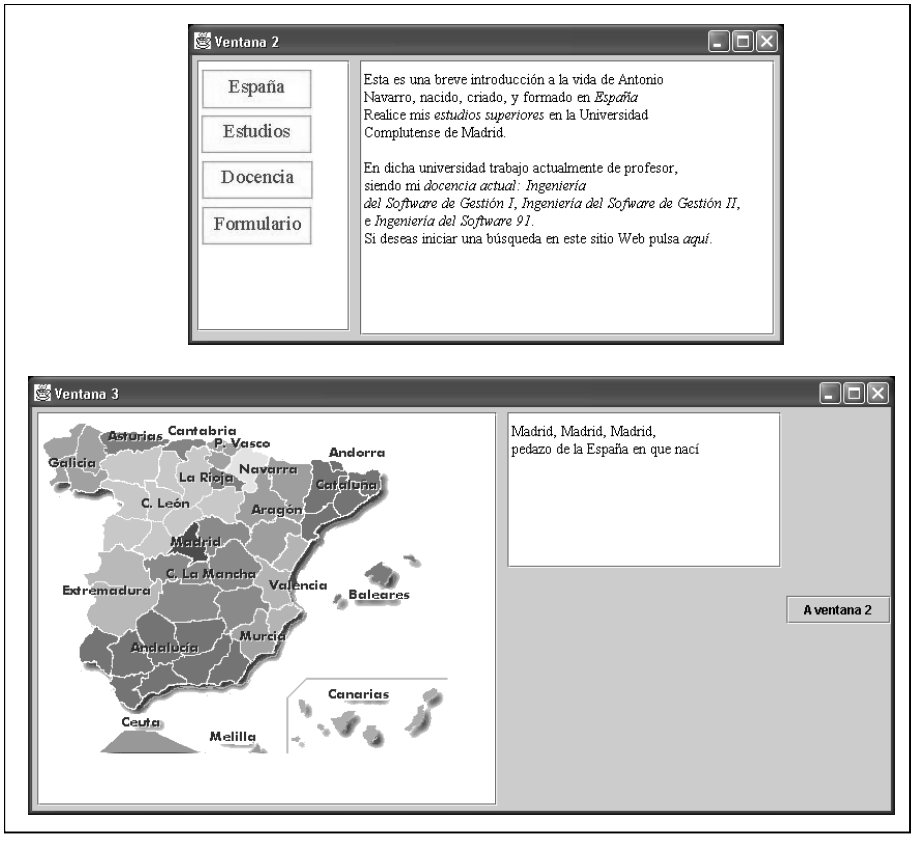

Figura 6.15 Ventanas generadas durante la navegación por la aplicación

La tercera interfaz del ejemplo viene representada por la Figura 6.16.

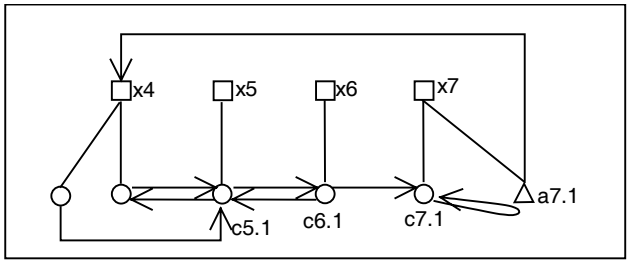

Figura 6.16 Interfaz 3

Su parte del documento de la aplicación es la siguiente.

```
<ventana id="v5">
     <nombre>Ventana 4</nombre>
     <panel id="p5.1" tam= "155, 265">
        <contenidoPanel>
           <contenidoDefecto>/profesor/índice</contenidoDefecto>
        </contenidoPanel>
        <destinoEnlaces panel="p6.1"/>
     </panel>
     <panel id="p5.2" tam="400, 265">
        <contenidoPanel>
           <contenidoDefecto>/profesor/biografía</contenidoDefecto>
        </contenidoPanel>
        <destinoEnlaces panel="p6.1"/>
     </panel>
  </ventana>

  <ventana id="v6">
    <nombre>Ventana 5</nombre>
    <panel id="p6.1">
      <contenidoPanel>
        <contenido>/profesor/españa</contenido>
        <contenido>/profesor/estudios</contenido>
        <contenido>/profesor/docencia</contenido>
        <contenido>/profesor/formulario</contenido>
      </contenidoPanel>
      <destinoEnlaces panel="p7.1"/>
    </panel>
  </ventana>

  <ventana id="v7">
    <nombre>Ventana 6</nombre>
    <panel id="p7.1">
       <contenidoPanel>
        <contenido>/profesor/resultado</contenido>
        <contenidoGrupo>/profesor/localidades</contenidoGrupo>
        <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
      </contenidoPanel>
      <destinoEnlaces panel= "p8.1"/>
    </panel>
  </ventana>

<ventana id="v8">
     <nombre>Ventana 7</nombre>
     <panel id="p8.1">
       <contenidoPanel>
       </contenidoPanel>
     </panel>
     <activadorV id="av8.1" ventana= "v5">
       <nombre>A ventana 4</nombre>
     </activadorV>
  </ventana>
```

Las ventanas generadas durante la interacción pueden verse en la Figura 6.17.

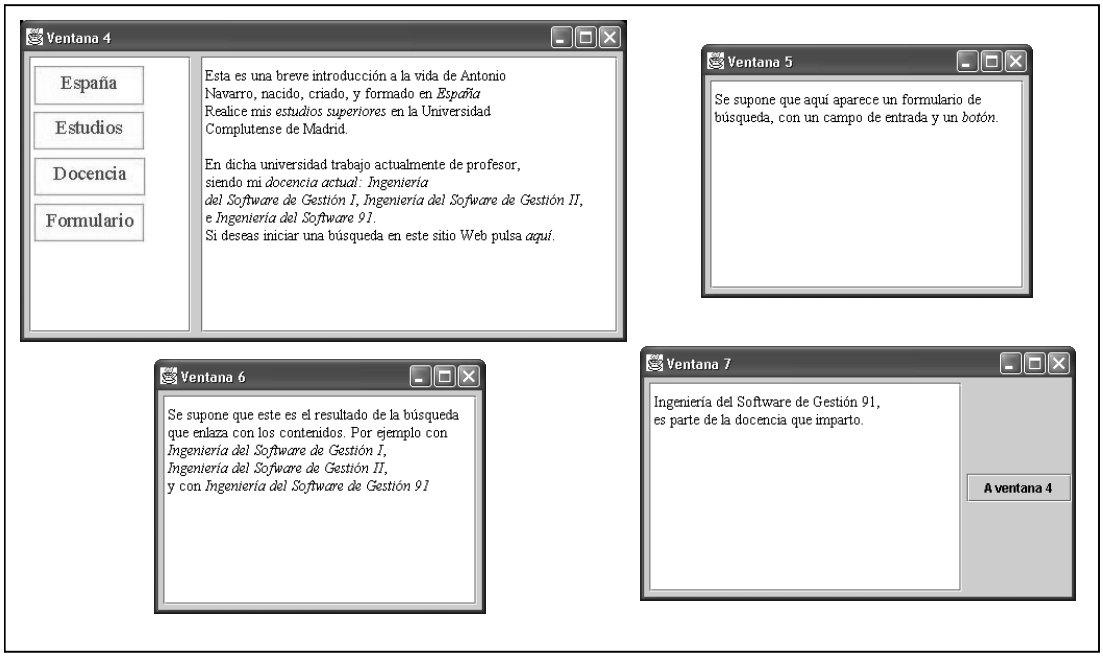

Figura 6.17 Ventanas generadas durante la navegación por la aplicación

La cuarta interfaz del ejemplo viene representada por la Figura 6.18.

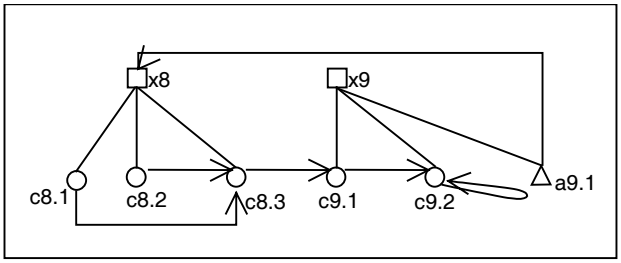

Figura 6.18 Interfaz 4

Su parte del documento de la aplicación es la siguiente.

```
<ventana id="v9">
     <nombre>Ventana 8</nombre>
     <panel id="p9.1" tam= "155, 265">
        <contenidoPanel>
           <contenidoDefecto>/profesor/índice</contenidoDefecto>
        </contenidoPanel>
        <destinoEnlaces panel="p9.3"/>
     </panel>
     <panel id="p9.2" tam="400, 265">
        <contenidoPanel>
           <contenidoDefecto>/profesor/biografía</contenidoDefecto>
        </contenidoPanel>
        <destinoEnlaces panel="p9.3"/>
     </panel>
     <panel id="p9.3" tam="400, 265">
      <contenidoPanel>
        <contenido>/profesor/españa</contenido>
        <contenido>/profesor/estudios</contenido>
        <contenido>/profesor/docencia</contenido>
        <contenido>/profesor/formulario</contenido>
       </contenidoPanel>
```

```
    <destinoEnlaces panel="p10.1"/>
  </panel>
</ventana>
<ventana id="v10">
  <nombre>Ventana 9</nombre>
  <panel id="p10.1">
     <contenidoPanel>
      <contenido>/profesor/resultado</contenido>
      <contenidoGrupo>/profesor/localidades</contenidoGrupo>
      <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
    </contenidoPanel>
    <destinoEnlaces panel= "p10.2"/>
  </panel>

  <panel id="p10.2">
     <contenidoPanel>
     </contenidoPanel>
  </panel>
  <activadorV id="av10.1" ventana= "v9">
     <nombre>A ventana 8</nombre>
   </activadorV>
</ventana>
```

La Figura 6.19 muestra las ventanas generadas.

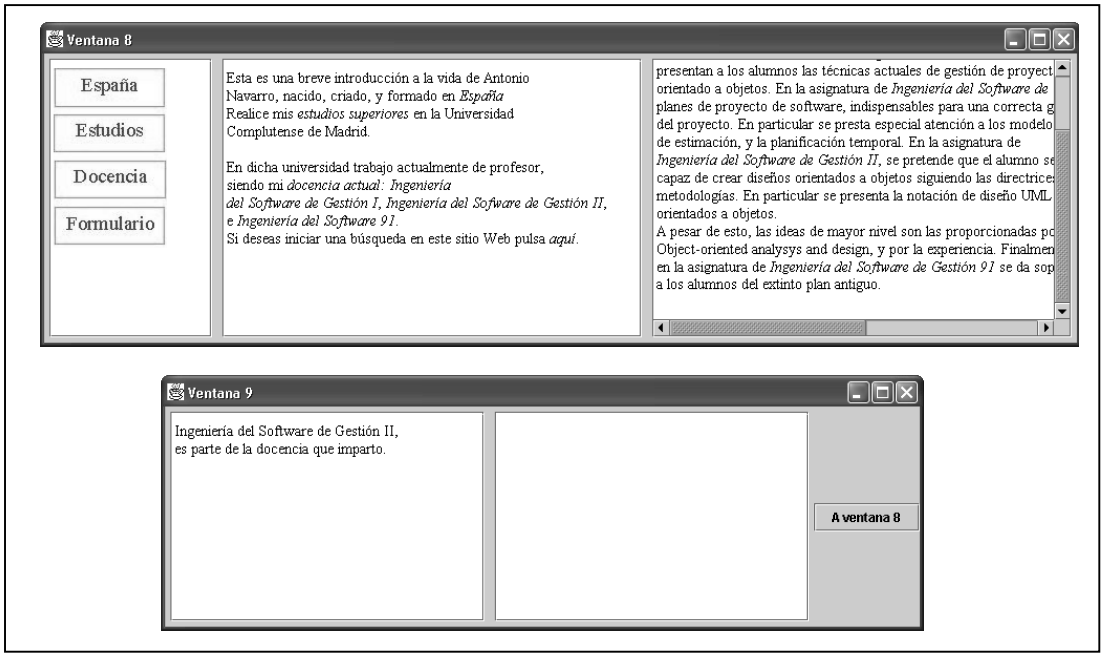

Figura 6.19 Ventanas generadas durante la navegación por la aplicación

Finalmente la Figura 6.20 representa la quinta interfaz del ejemplo.

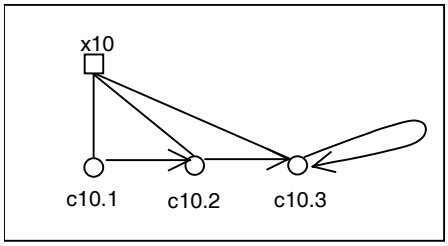

Figura 6.20 Interfaz 5

Cuya parte en la DTD de la aplicación es la siguiente.

```
<ventana id="v11">
     <nombre>Ventana 10</nombre>
     <panel id="p11.1"  tam= "300, 380">
        <contenidoPanel>
           <contenidoDefecto>/profesor/biografía</contenidoDefecto>
        </contenidoPanel>
        <destinoEnlaces panel="p11.2"/>
     </panel>
     <panel id="p11.2" tam="445, 380">
        <contenidoPanel>
          <contenido>/profesor/españa</contenido>
           <contenido>/profesor/estudios</contenido>
           <contenido>/profesor/docencia</contenido>
          <contenido>/profesor/formulario</contenido>
         </contenidoPanel>
      <destinoEnlaces panel="p11.3"/>
    </panel>
    <panel id="p11.3" tam="200, 380">
       <contenidoPanel>
          <contenido>/profesor/resultado</contenido>
           <contenidoGrupo>/profesor/localidades</contenidoGrupo>
           <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
       </contenidoPanel>
       <destinoEnlaces panel="p11.3"/>
    </panel>

  </ventana>
```

La ventana generada durante la navegación puede verse en la Figura 6.21.

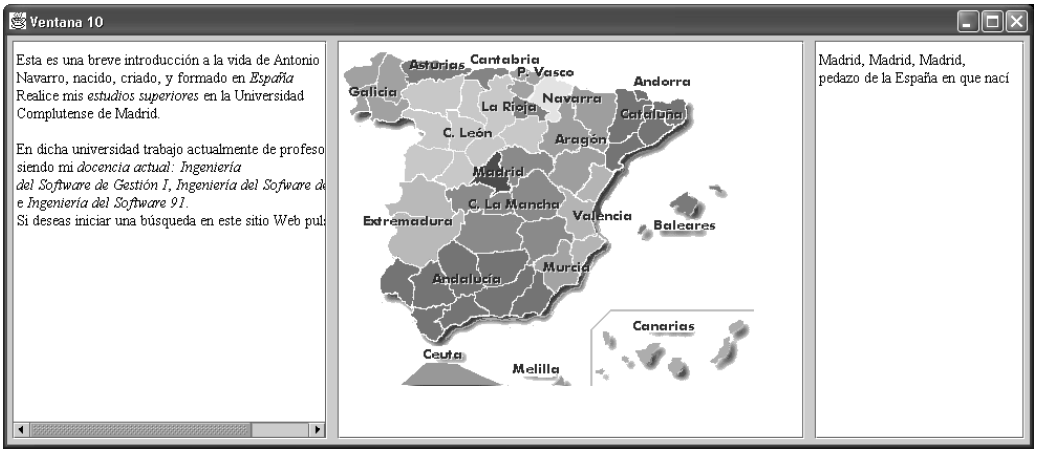

Figura 6.21 Ventana generada

6.3.6 PlumbingMatic

Si bien es posible gestionar el modelo de proceso PlumbingXJ de una forma "manual", sería deseable la creación de un entorno integrado de diseño y desarrollo, el entorno PlumbingMatic.

De esta forma se podría gestionar la creación de la DTD de contenidos, garantizando las restricciones necesarias, y facilitando a su vez la creación de sus instancias, así como la el documento de la aplicación. Además, proporcionaría el entorno ideal para el sobremarcado, garantizando la validez de las expresiones XPath/XSLT.

Dicho entorno también sería capaz de construir el documento de diseño, utilizando una interfaz gráfica basada en los diagramas Pipe que permitiese generar de manera automática el documento de diseño XML. Además, ahora GAP sería parte integrante de PlumbingMatic, permitiéndose la generación de prototipos "directamente" a partir de las representaciones Pipe de la aplicación. De esta forma la herramienta proporcionaría tres vistas de la aplicación equivalentes:

- El modelo Pipe
- El documento de la aplicación/diseño
- El prototipo

Si también se incluyese una vista *gráfica* formada por ventanas, paneles y activadores según su representación real, podría construirse la aplicación de la misma forma que con una herramienta visual. Este hecho podría facilitar la posible aplicación de PlumbingXJ por parte del cliente directamente.

La construcción de PlumbingMatic queda para trabajo futuro, siendo GAP una implementación de (a nuestro juicio) la parte más importante de dicha herramienta CASE.

6.4 Conclusiones

La naturaleza específica de las aplicaciones hipermedia demanda prestar especial atención a las fases de conceptualización y prototipado de este tipo de aplicaciones [Fraternali99], [Ginige, 97]. Los sistemas de representación hipermedia clásicos no cumplen con éxito esta misión debido a su concepción original como herramienta de diseño, y por su pobre soporte CASE [Barry 01]. En este capítulo hemos introducido el modelo hipermedia Pipe (específicamente concebido como herramienta de conceptualización) dentro del modelo de proceso de Fraternali-Ginige/Lowe para obtener el modelo de proceso Plumbing. Plumbing se ve beneficiado de la alta abstracción de Pipe, el cual permite diversas técnicas concretas de prototipado, las cuales pueden determinar en mayor o en menor medida las técnicas de diseño e implementación final.

Concretando las técnicas de prototipado hemos obtenido el modelo de proceso PlumbingXJ, el cual utiliza XML y Java para construir prototipos de manera automática basados en la semántica Pipe (para aplicaciones estáticas por el momento). En efecto, ahora las estructuras Pipe se codifican en XML sirviendo de entrada para GAP, el cual construye de manera automática los prototipos de la aplicación. Las ventajas que se derivan de esta aproximación son varias. En primer lugar, la conceptualización Pipe es una primera herramienta que se utiliza como representación de alto nivel de la aplicación hipermedia. Basándose en esta representación, los desarrolladores son capaces de construir los documentos de la aplicación y de contenidos, ambos en formato XML. La utilización de XML favorece la comunicación entre clientes y desarrolladores, ya que debido a su alto nivel de declaratividad, los clientes ven facilitada la tarea de proporcionar tanto los contenidos como el esquema navegacional de la aplicación. Además utilizando XML los clientes pueden proporcionar una estructura a sus contenidos, algo imprescindible en ciertos dominios, como el educativo [Navarro 00b]. Pero no solo los clientes se benefician de esta declaratividad. Los desarrolladores pueden construir prototipos de aplicaciones sin necesidad de escribir código (salvo para los procesos subordinados, si se desea incluirlos). Solamente deben construir o modificar el documento de la aplicación para generar la aplicación hipermedia. De esta forma se evitan la tarea de codificar a mano cada cambio en los contenidos o en el esquema navegacional de la misma. No debemos olvidarnos tampoco de la ventaja derivada de separar los contenidos del esquema navegacional que se les va a dar. Ahora podemos proporcionar distintas interfaces a los mismos contenidos (como en el ejemplo del sitio Web), o al revés proporcionar el mismo interfaz a distintos contenidos. En efecto si en el ejemplo del sitio Web optamos por cambiar el mapa de España, por un texto con anclas hacia Madrid y Lepe, solo son necesarios cambios en el documento de contenidos, ya que mientras mantengamos el nombre de dicho contenido la referencia XPath incluida en el documento de la aplicación va a seguir siendo válida.

Por último comentar que tanto la representación Pipe de la aplicación, como los documentos de contenidos y de aplicación, no comprometen en ninguna medida el diseño e implementación final de la misma. En efecto, las representaciones Pipe son los suficientemente abstractas como para incluir las representaciones proporcionadas por otros modelos (como los relacionales, por ejemplo). Además la existencia de datos en formato XML es una garantía de compatibilidad y accesibilidad por parte de diversas aplicaciones, ya que en la actualidad existen sistemas de intercambio basados en XML.

7 Casos prácticos

En este capítulo veremos varios ejemplos de aplicación de Pipe y PlumbingXJ. Tres de ellos están centrados en el dominio educativo, área de especial interés por parte de nuestro grupo de investigación [Fernández 97], [Fernández-Valmayor 99], [Navarro 00b]. A pesar de que los desarrollos reales no se hicieron teniendo en cuenta las ideas aquí propuestas (de hecho, las ideas aquí propuestas son en parte fruto de estos desarrollos), haremos un ejercicio de ingeniería inversa para modelar a posteriori dichas aplicaciones. En particular aplicaremos nuestras técnicas a *Lire en Français* (*LeF*), una herramienta en CD-ROM para la comprensión de textos en lengua francesa; a un sitio Web que contiene un conjunto de tutoriales sobre XML y tecnologías relacionadas; y finalmente a *eAula* un sistema Web de e-learning. Fuera del dominio educativo aplicaremos nuestra aproximación al modelado de la Web de Amazon, rica en enlaces y contenidos dinámicamente generados.

7.1 Lire en Français

7.1.1 Introducción

Esta aplicación forma parte del Proyecto Galatea y ha sido realizada en el marco del programa SÓCRATES-LINGUA de la Unión Europea (acción D) [Fernández-Valmayor 99]. Dicha herramienta es un método interactivo de autoaprendizaje que permite comprender textos en lengua francesa. Está dirigido tanto a principiantes como para lectores avanzados. Incluso un lector sin conocimientos previos puede alcanzar un buen primer nivel de comprensión, y cualquier lector con conocimientos básicos en lengua francesa perfeccionará al máximo su capacidad lectora. La metodología docente está basada en un método de autoaprendizaje interactivo que no propone únicamente una valoración tradicional del trabajo del lector; en efecto, éste puede repetir cada una de las actividades cuantas veces le sea necesario y resolver él solo con el interfaz todos los problemas que se encuentre. Cada actividad contiene su propia evaluación y el lector puede fácilmente evaluar, a lo largo de diferentes recorridos, la progresión de su comprensión en la lectura del texto. Por ello, este método le invita e instiga a que utilice libremente todas las herramientas que tiene a su disposición en cada texto, pudiendo ir, en cualquier momento, de las actividades de comprensión global a las de comprensión pormenorizada.

Aunque originalmente esta herramienta fue concebida para su difusión en CD-ROM, en la actualidad estamos desarrollando una versión para la Web (http://www.simba.vsf.es/IndiceDemos.htm) [Navarro 01a]. Como ya hemos comentado el desarrollo de esta aplicación se hizo con independencia de las técnicas propuestas en esta tesis. De hecho, las técnicas propuestas en esta tesis surgieron en gran parte como respuesta a los problemas de diseño, desarrollo y mantenimiento que presentó la construcción de LeF. La ventaja de la ingeniería inversa es que el diseño parte de una especificación de requisitos total, limitándose a una mera representación del conocimiento del que ya se dispone. Utilizaremos una especificación de requisitos en lenguaje natural, bastante lejana a las técnicas estructuradas propuestas por [Sommerville 01]. No es falta de aproximación formal la causa principal de esta decisión, sino las características intrínsecas de las aplicaciones hipermedia [Fraternali 99] que nos lleva a buscar otras técnicas de especificación. Precisamente las notaciones inducidas por los sistemas de representación hipermedia, y en particular la propuesta por el modelo Pipe, se muestran como la herramienta idónea para capturar especificaciones. La ventaja adicional de Pipe es que estas especificaciones son más independientes de diseños concretos que las proporcionadas por otros sistemas de representación hipermedia.

Procederemos pues a la representación Pipe de LeF. En vez de optar por un listado inicial de especificaciones y el modelo Pipe correspondiente, iremos considerando especificaciones de partes de LeF para luego proporcionar su representación Pipe.

7.1.2 Análisis de requisitos y conceptualización

Como estamos haciendo ingeniería inversa, y está bien claro que es lo que debe hacer la herramienta, tomaremos la información proporcionada en la introducción como el primer paso de análisis de requisitos. Además, obviaremos algunos detalles de la implementación final con el fin de aligerar un poco el diseño.

En cuanto a la conceptualización empecemos describiendo el esquema navegacional. En esta aplicación se optó por elegir un esquema navegacional arbóreo, de tal forma que el alumno tuviese acceso en todo

momento al padre y a sus hermanos del árbol. Este esquema surgió después de varios prototipos, en los que se vio que una estructura menos jerárquica producía problemas de desorientación en los alumnos. De esta forma, la primera ventana de la aplicación está formada por una foto de la Torre Eiffel y cinco frases que dan el acceso a otros cinco respectivos textos: *Versailles*, *Le mont Saint-Michel*, *Toute la magie de la piste*, *L´Incendie d´un appartament*, y *Paul Gaugin*. Como el esquema navegacional que se va a imponer sobre todos ellos es el mismo, nos centraremos solamente en uno, por ejemplo *L´Incendie d´un appartament*. S. Una vez que seleccionamos alguno de estos textos aparece una ventana con la foto de un quiosco, que nos permite acceder a los distintos recorridos posibles que el estudiante puede hacer sobre el texto: *Recorrido Guiado, Recorrido Libre, Para escuchar*. También es posible volver a la ventana principal. La Figura 7.1 recoge esta información sobre las dos primeras ventanas de LeF.

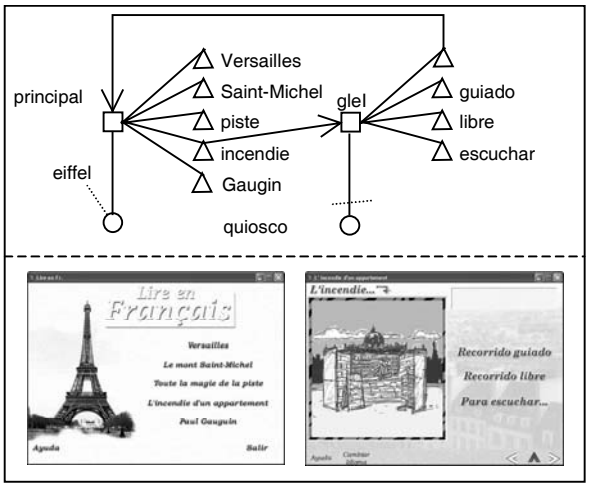

Figura 7.1 Ventana principal de LeF y ventana del texto. Hemos obviado algunos enlaces con origen en activadores de ventanas

Si el usuario selecciona el recorrido guiado aparece una ventana donde se muestra una hoja de periódico y los cinco posibles recorridos que el estudiante puede llevar a cabo: *Este texto es...*, *Se trata de...*, *¿Qué quiere decir...?*, *¿Quién, cuando, donde?*, *Y para acabar de comprender*. También aparece un botón que permite acceder a la ventana **glel**. La Figura 7.2 muestra dicha ventana.

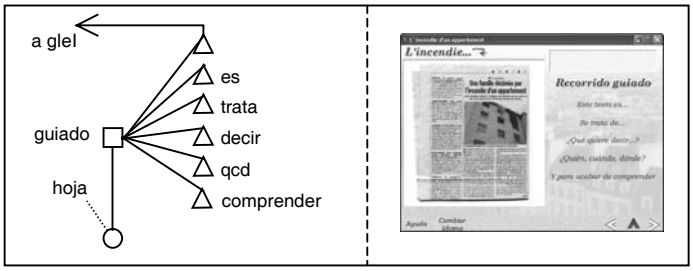

Figura 7.2 Ventana del *Recorrido guiado*.

Si el estudiante selecciona la opción de *Este texto es* (textoEs), donde tiene acceso tres ventanas consecutivas que intentan hacer comprender al usuario que tipo de texto está leyendo (*periodístico* en este caso). La Figura 7.3 ilustra la representación Pipe de estas ventanas.

Nótese como en LeF la navegación está encomendada en su mayor parte a los activadores de ventanas. Nótese también como los contenidos tipoTextoI, identificarI y arrastrarI se corresponden con procesos subordinados. En este caso son ejercicios que evalúan los conocimientos del alumno. En el nivel de conceptualización hipermedia no tienen ninguna representación significativa. Como hemos comentado en capítulos anteriores, en la fase de diseño deben proporcionarse algún mecanismo de representación de los mismos (diagramas UML, por ejemplo).

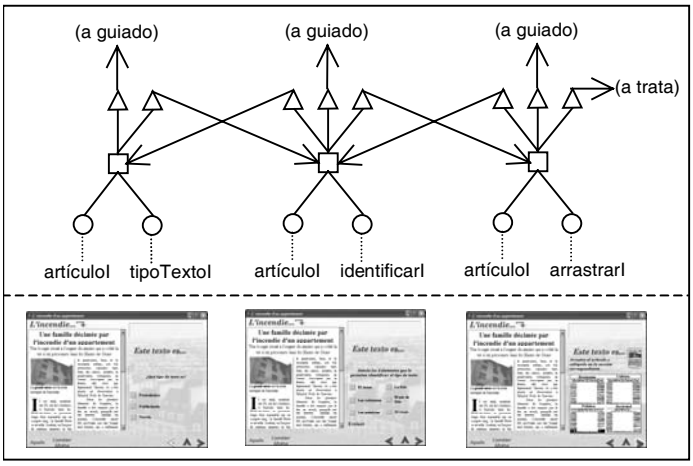

Figura 7.3 Ventanas de *Este texto es...*

Si hubiese elegido el recorrido *Se trata de...* (trata) el estudiante hubiese tenido acceso a dos ventanas consecutivas que exploran la comprensión del texto por parte del alumno. La Figura 7.4 ilustra estas ventanas.

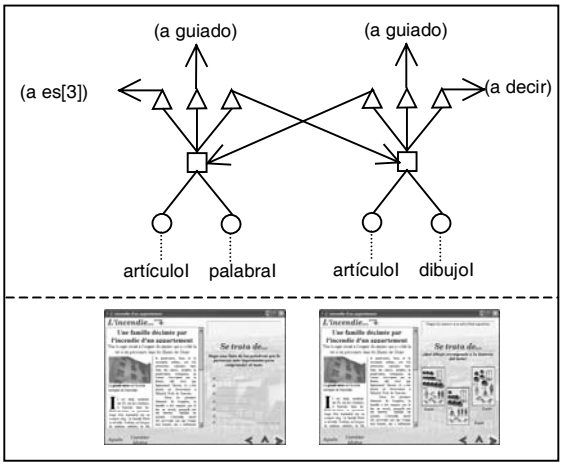

Figura 7.4 Ventanas de *Se trata de ...*

Al igual que antes los contenidos palabraI y dibujoI se corresponden con procesos subordinados. Si hubiese elegido *¿Qué quiere decir?* (decir) el alumno hubiese accedido a seis ventanas consecutivas que intentan mejorar su vocabulario en francés. La Figura 7.5 recoge estas ventanas.

Podemos comprobar, en la tónica general de la aplicación, que la navegación sigue encomendada a los activadores de ventana de la misma. Además vemos como siempre uno de los contenidos es un proceso subordinado. En cuanto a la presentación simultánea en una misma ventana de dos contenidos (el artículo, y el proceso) necesitamos enlaces binarios entre la lista de palabras (listPal) y el resto de contenidos. La Figura 7.6 ilustra el grafo de contenidos.

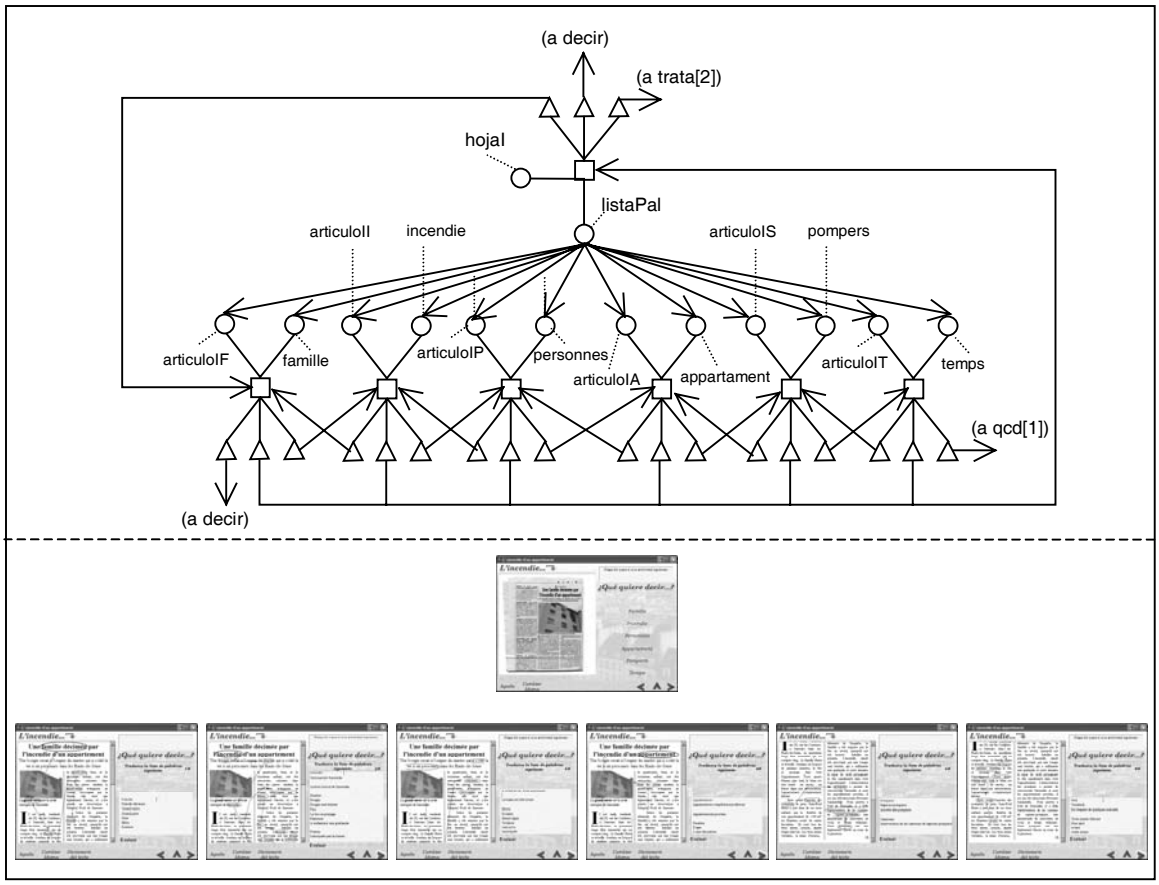

Figura 7.5 Ventanas de *¿Qué quiere decir...?*

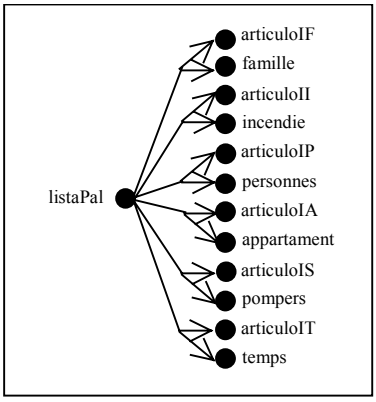

Figura 7.6 Grafo de contenidos para *¿Qué quiere decir?* Nótese los enlaces binarios entre *listaPal* y el resto de contenidos.

La selección de *¿Quién, cuando, donde?* en el *Recorrido Guiado* nos da acceso a tres ventanas consecutivas que sirven para fijar los protagonistas de la historia, y el marco espacio-temporal de la misma. La Figura 7.7 muestra estas ventanas.

Finalmente, *Para acabar de comprender* da acceso a once ventanas, seis en francés, y cinco en español que evalúan el nivel de comprensión del texto por parte del alumno. La Figura 7.8 ilustra este apartado.

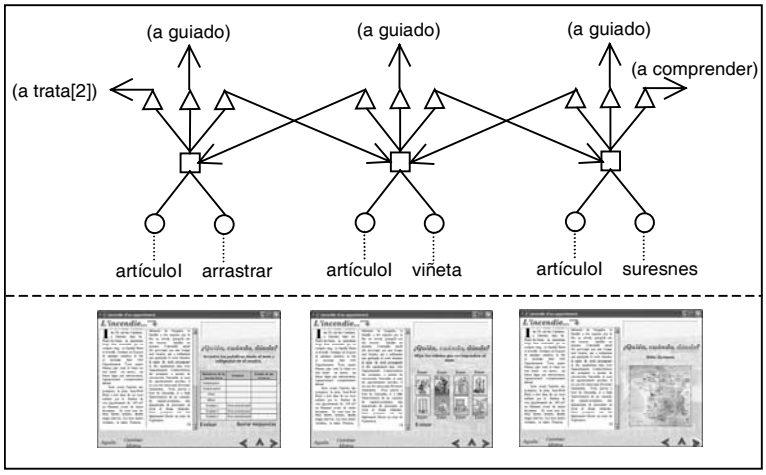

Figura 7.7 Ventanas de *¿Quién, cuando, donde?*

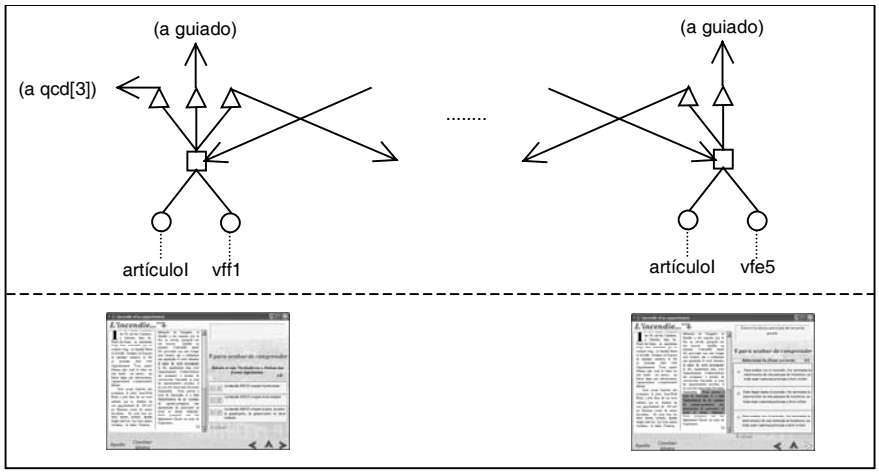

Figura 7.8 Ventanas de *Para acabar de comprender*. Hemos obviado nueve pantallas por su similitud con el esquema navegacional de los apartados previos.

Cómo hemos podido comprobar, en el recorrido libre existe un esquema navegacional bastante homogéneo, estando limitados los contenidos a un conjunto de elementos inconexos, en su mayoría (salvo en el apartado *¿Qué quiere decir?*). Además un gran número de estos contenidos son procesos subordinados. La razón es bastante obvia. Se trata ejercicios que constantemente evalúan los conocimientos del alumno, pero que en ningún caso afectan a la navegación.

En el apartado del *Contenido Libre* nos encontramos en una situación bastante distinta. Básicamente es una ventana donde el alumno puede seleccionar frases de un texto, y palabras de dichas frases para comprobar el significado contextual de dicha palabra. En este caso, el grafo de contenidos puede considerarse de dos formas distintas. La primera es considerar que las frases enlazan de manera dinámica con la definición de sus palabras. Es decir, hay una aplicación que en base al ancla devuelve la definición de la palabra utilizando un diccionario a modo de base de datos. La segunda es considerar al diccionario como un contenido direccionable, enlazando las palabras de la frase de manera estática con posiciones concretas dentro del diccionario (que por supuesto se corresponden con las definiciones buscadas). La Figura 7.9 ilustra estos supuestos.

Como la aplicación final sigue el esquema dinámico, vamos a utilizar esta en el modelo. El esquema navegacional que se va a imponer a estos contenidos es bastante sencillo. Se van a presentar todos los contenidos en una misma ventana. El texto en un panel. Las frases seleccionadas en otro, y finalmente las definiciones en un tercer panel. La Figura 7.10 muestra este esquema navegacional, la asignación de contenidos y la canalización de enlaces.

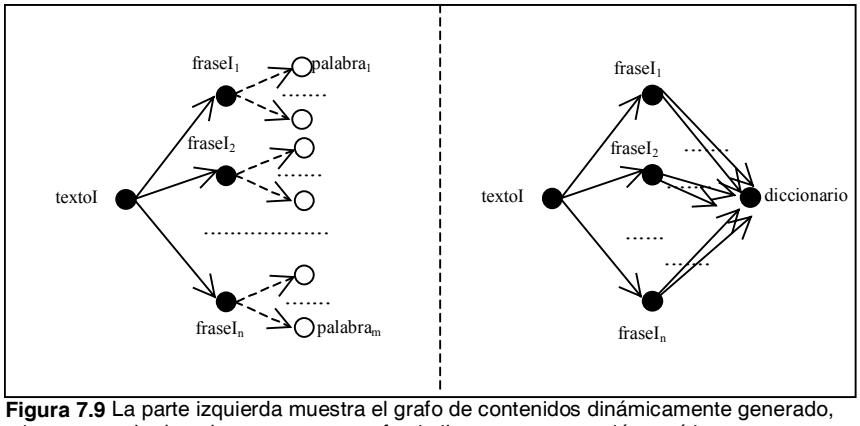

Figura 7.9 La parte izquierda muestra el grafo de contenidos dinámicamente generado, mientras que la derecha muestra un grafo similar en su concepción estática.

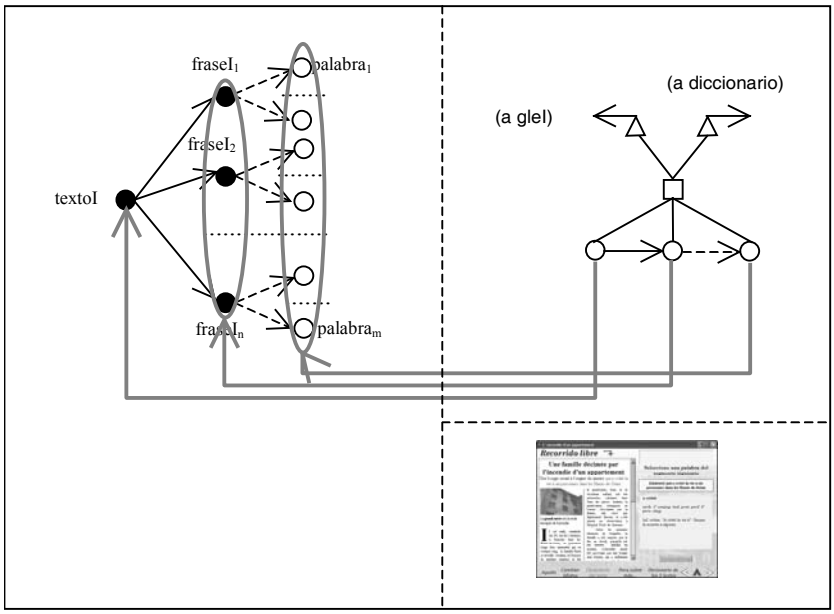

Figura 7.10 Esquema navegacional y relación de este con el grafo de contenidos. Las flechas en línea gruesa representan la asignación de contenidos (función *d*). La tubería en trazo sólido canaliza a los enlaces de contenidos estáticos (función *l*). La tubería en traza discontinua canaliza los enlaces de contenidos dinámicos (función *l*).

Como podemos ver en el esquema navegacional descrito en la Figura 7.10, hay un botón que da acceso a una ventana diccionario. Esta ventana contiene un diccionario con las palabras que aparecen en los cinco textos de la aplicación. Además existe la posibilidad de acceder a las definiciones mediante un formulario de búsqueda. Al igual que en la Figura 7.9 ahora tenemos la opción de representar los enlaces entre las palabras de la aplicación y los diccionarios de manera dinámica o estática. En ambos casos la búsqueda es dinámica. La Figura 7.11 ilustra esta posibilidad.

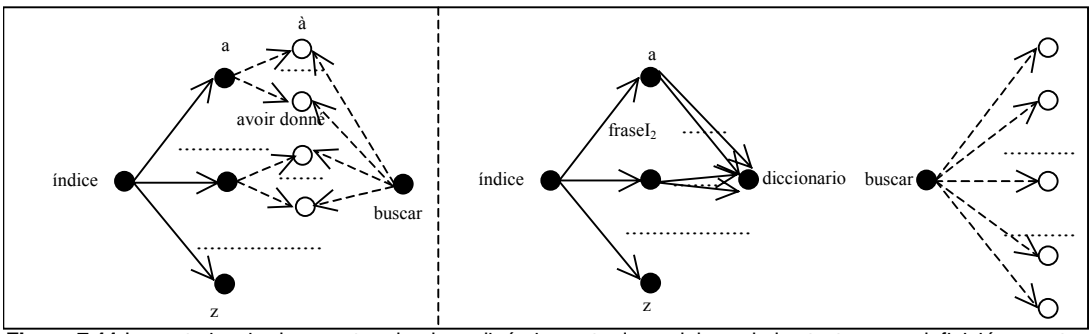

Figura 7.11 La parte izquierda muestra el enlace dinámico entre las palabras de los textos y su definición, y entre el formulario de búsqueda y las definiciones encontradas. La parte derecha muestra el enlace estático entre las palabras de los textos y su definición, y el enlace dinámico entre el formulario de búsqueda y las definiciones encontradas.

También al igual que antes, nos vamos a quedar con la representación dinámica de estos contenidos (parte izquierda de la Figura 7.11). El esquema navegacional impuesto estará formado por una ventana con cuatro paneles. En uno se muestra el índice alfabético. En otro se muestran las palabras de los cinco textos. En otro el formulario de búsqueda. En el cuarto se muestran las definiciones. La Figura 7.12 representa este esquema, la asignación de contenidos, y la canalización de enlaces.

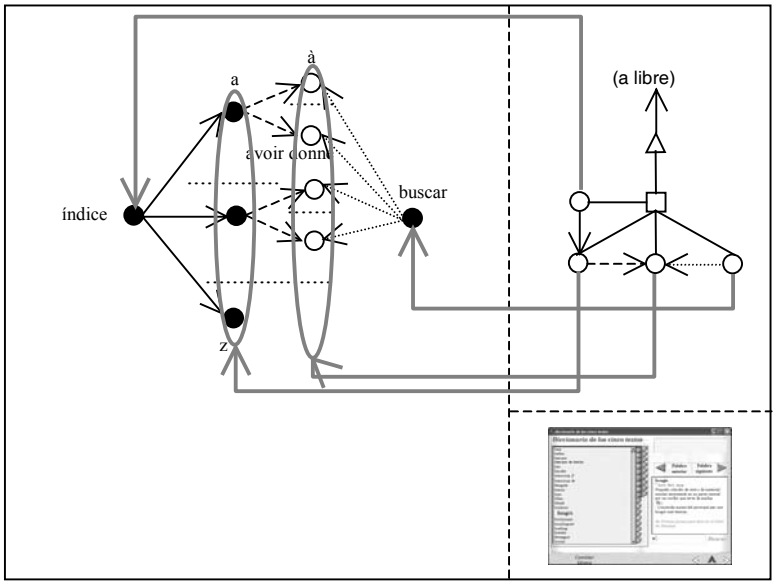

Figura 7.12 Esquema navegacional del diccionario y su relación con los contenidos. Las flechas en trazo grueso muestran la asignación de contenidos (función *d*). La tubería en trazo continuo canaliza los enlaces entre el índice y las palabras. La tubería en trazo discontinuo canaliza los enlaces entre la lista de palabras y sus definiciones. La tubería punteada canaliza los enlaces entre el formulario de búsqueda y las definiciones.

Finalmente solo nos queda modelar la tercera opción que tiene el alumno, además de los recorridos libre y guiado, *Para escuchar...*. Este módulo está formado por un texto con frases seleccionables de tal forma que al pinchar una frase podemos oír su pronunciación en francés a la vez que la frase cambia de color (es decir, tenemos un *karaoke*). La Figura 7.13 muestra dichos contenidos.

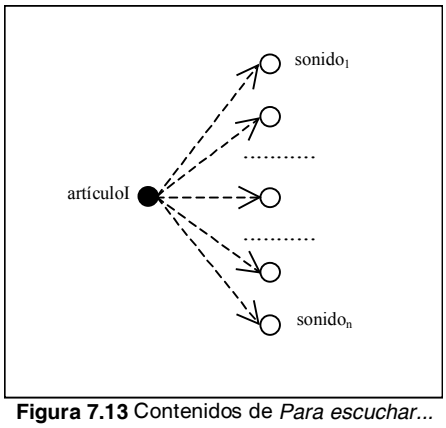

Figura 7.13 Contenidos de *Para escuchar...*

El esquema navegacional que se le va a imponer consiste en una ventana con dos paneles. En uno aparece el texto, y en otro las frases. La Figura 7.14 ilustra este esquema.

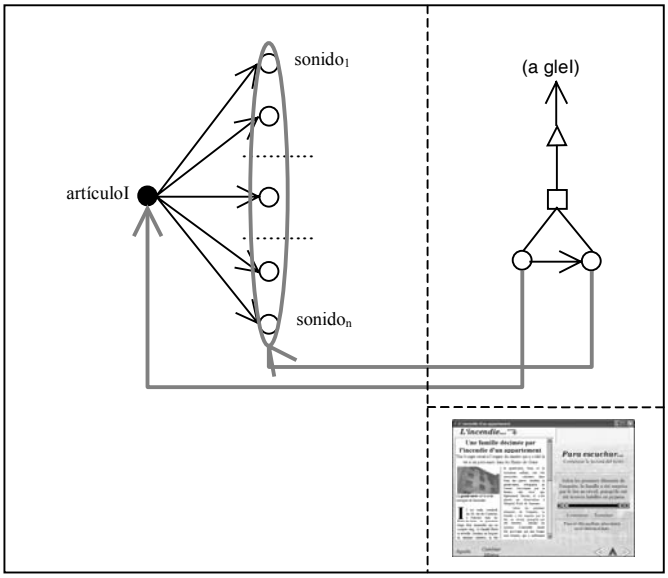

Figura 7.14 Esquema navegacional de *Para escuchar...* y relación con los contenidos. Las líneas en trazo grueso muestran la asignación de contenidos (función *d*). La única tubería canaliza todos los enlaces de estos contenidos (función *l*).

7.1.3 Prototipado

En un ejercicio de ingeniería inversa no tiene demasiado sentido llevar a cabo esta fase. De todas formas veremos como es el prototipo para la el caso del diccionario contextual. A pesar de haber elegido una caracterización dinámica de los enlaces entre las palabras y su definición para hacer posible un prototipado más rápido y expresivo supondremos un enlace estático entre estos contenidos. En ningún caso esta decisión supone una inversión desperdiciada en tiempo, ya que el propio diccionario que se utiliza para generar las definiciones dinámicamente está representado en XML. La cuestión es que los clientes querían una definición "tan contextual" de cada expresión que no fue suficiente la aproximación presentada en la Figura 7.9. En su lugar se optó por proporcionar una definición ad-hoc para cada expresión. La Figura 7.15 recoge esta nueva aproximación.

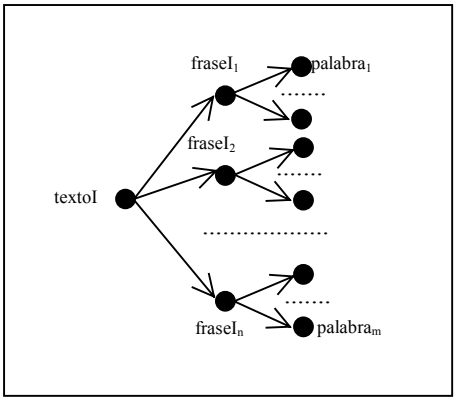

Figura 7.15 Grafo de contenidos real para los elementos del diccionario contextual

La la siguiente es la dtd de contenidos acorde con la Figura 7.15.

```
<!ELEMENT contenidos (artículo, frases, definiciones)>

<!ELEMENT artículo (ancla+)>
<!ATTLIST artículo imagen CDATA #REQUIRED>
<!ELEMENT ancla EMPTY>
<!ATTLIST ancla href CDATA #REQUIRED coords CDATA #REQUIRED>
```

```
<!ELEMENT frases (frase+)>
<!ELEMENT frase (palabra+)>
<!ATTLIST frase id ID #REQUIRED>
<!ELEMENT palabra (#PCDATA)>
<!ATTLIST palabra href IDREF #REQUIRED>

<!ELEMENT definiciones  (definición+)>
<!ELEMENT definición (pal, morfología, significado?)>
<!ATTLIST definición id ID #REQUIRED>
<!ELEMENT pal (#PCDATA)>
<!ELEMENT morfología  (#PCDATA)>
<!ELEMENT significado  (#PCDATA)>
```

La instancia de dicha DTD puede verse en el Apéndice C. Como se puede comprobar en dicha instancia solo hemos considerado un par de frases del texto. El documento de la aplicación para el esquema navegacional de la Figura 7.10 es el siguiente.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="ap4">

   <ventana id= "v1">
      <nombre>ventana 1</nombre>

      <panel id= "p1.1" tam="350, 405">
        <contenidoPanel>
         <contenidoDefecto>/contenidos/artículo</contenidoDefecto>
        </contenidoPanel>
          <destinoEnlaces panel="p1.2"/>
      </panel>

      <panel id= "p1.2" tam="350, 100">
        <contenidoPanel>
         <contenidoGrupo>/contenidos/frases</contenidoGrupo>
        </contenidoPanel>
         <destinoEnlaces panel="p1.3"/>
      </panel>

      <panel id= "p1.3" tam="200, 200">
        <contenidoPanel>
         <contenidoGrupo>/contenidos/definiciones</contenidoGrupo>
        </contenidoPanel>
      </panel>
   </ventana>

</aplicacion>
```

Finalmente, la Figura 7.16 muestra el prototipo generado.

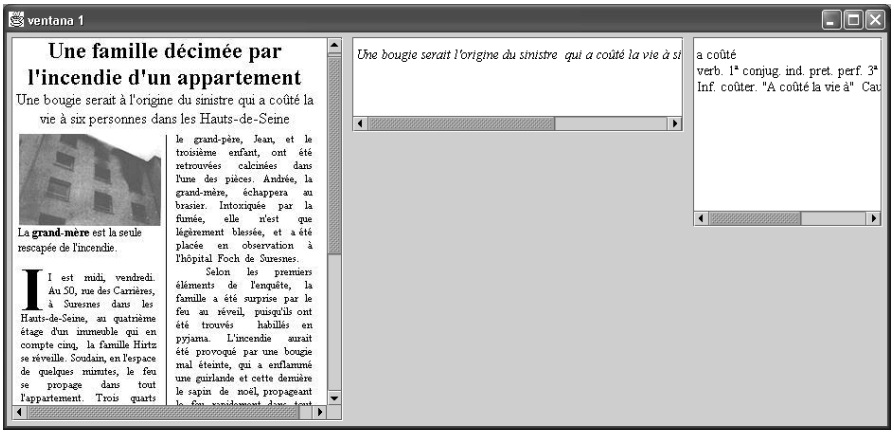

Figura 7.16 Prototipo GAP

7.2 Tutoriales XML

7.2.1 Introducción

Esta aplicación hipermedia (*http://www.simba.vsf.es/FrameDoc.htm*) está formada por un conjunto de tutoriales on-line sobre XML y tecnologías relacionadas (SAX, DOM, XSLT, y XHMTL). En este caso hemos preferido obviar los contenidos multimedia en aras de un mejor rendimiento [Navarro 01a].

7.2.2 Análisis de requisitos y conceptualización

En este caso los contenidos van a estar formados por un índice general que da acceso cinco índices acerca de los tópicos descritos por los tutoriales, y de forma simultánea al primer contenido de cada uno de estos temas (es decir, tenemos un enlace binario). Los índices de cada tópico dan acceso a los contenidos y también a glosarios de cada tema. Estos glosarios contienen enlaces a las principales páginas donde aparecen ejemplos de las palabras que definen. Aunque en la práctica estos glosarios se generan a partir de los contenidos, a la hora de modelarlos este hecho es irrelevante, ya que dicha generación no depende de la activación de ningún hiperenlace. La Figura 7.17 muestra el grafo de contenidos de esta aplicación.

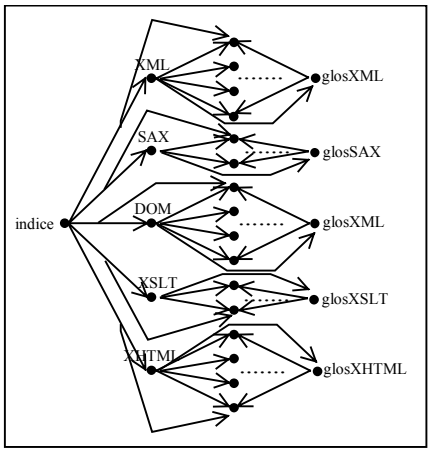

Figura 7.17 Grafo de contenidos de la aplicación.

El esquema navegacional a imponer será la típica estructura de marco HTML. En particular utilizaremos tres marcos. Uno para el índice general, otro para los índices de cada tema, y el último para los contenidos y los glosarios. El esquema navegacional queda recogido en la Figura 7.18. En esta figura hemos obviado la presencia de más enlaces entre cada índice de tópico y los contenidos. En la aplicación real se corresponden con un índice desplegable. Aunque el modelo Pipe no contempla modelar este comportamiento, si que se deberían haber incluido todos los enlaces correspondientes al máximo nivel de despliegue del índice. Se ha decidido no hacerlo así para mantener más simple la Figura 7.17.

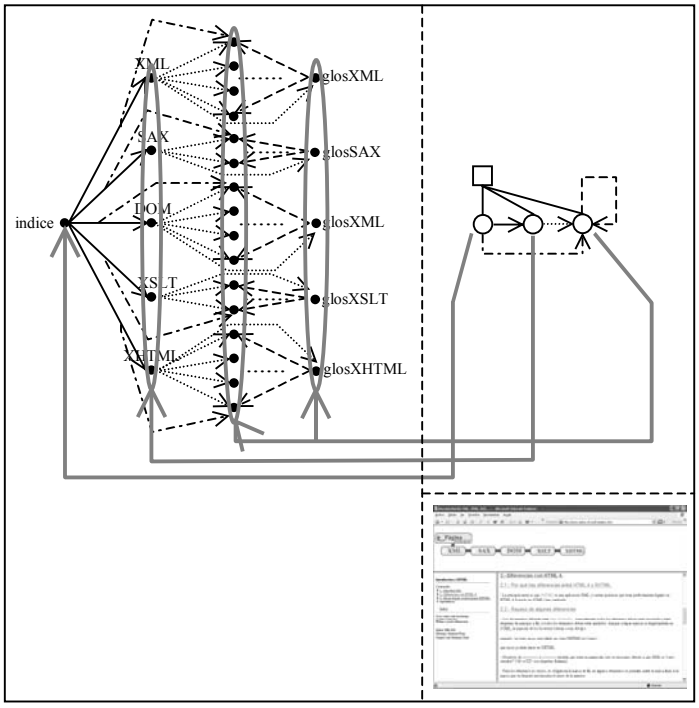

Figura 7.18 Esquema navegacional de la aplicación y su relación con el grafo de contenidos. Las líneas grises representan la asignación de contenidos (función *d*). Las tuberías canalizan los enlaces según el trazo correspondiente. Nótese que en esta ocasión ningún trazo denota enlaces generados dinámicamente

7.2.3 Prototipado

En un ejercicio de ingeniería inversa no tiene demasiado sentido llevar a cabo esta fase. De todas formas veremos como es el prototipo para la el caso de los contenidos XML. Ya que los glosarios se generan dinámicamente hemos decidido obviarlos.

La siguiente es la DTD de contenidos que hemos considerado.

```
<!ELEMENT contenidos (índice, tema+)>

<!ELEMENT índice (ancla+)>
<!ATTLIST índice imagen CDATA #REQUIRED>
<!ELEMENT ancla EMPTY>
<!ATTLIST ancla  href CDATA #REQUIRED
                 coords CDATA #REQUIRED>

<!ELEMENT tema (subÍndice, contenido)*>
<!ELEMENT subÍndice (#PCDATA|entrada)*>
<!ATTLIST subÍndice  id ID #REQUIRED>
<!ELEMENT entrada (#PCDATA)>
<!ATTLIST entrada href CDATA #REQUIRED>
<!ELEMENT contenido (#PCDATA|destino)*>
<!ATTLIST contenido id ID #REQUIRED>
<!ELEMENT destino (#PCDATA)>
<!ATTLIST destino name ID #REQUIRED>
```

Aunque unos contenidos de la forma (`índice`, `subÍndices`, `temas`) se hubiera ajustado mejor al esquema navegacional, hemos optado por mantener la anterior ya que captura mejor la estructura de los contenidos con independencia del tratamiento navegacional que se les vaya a dar. El documento de contenidos puede encontrarse en el Apéndice C.

El documento de la aplicación que representa a la Figura 7.18 es el siguiente.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="tutorialesXML">

   <ventana id= "v1">
      <nombre>ventana 1</nombre>

      <panel id= "p1.1" tam=" 140, 320">
        <contenidoPanel>
         <contenidoDefecto>/contenidos/índice</contenidoDefecto>
        </contenidoPanel>
          <destinoEnlaces panel="p1.2"/>
      </panel>

      <panel id= "p1.2" tam="320, 200">
        <contenidoPanel>
         <contenido>/contenidos/tema[1]/subÍndice</contenido>
         <contenido>/contenidos/tema[2]/subÍndice</contenido>
         <contenido>/contenidos/tema[3]/subÍndice</contenido>
         <contenido>/contenidos/tema[4]/subÍndice</contenido>
         <contenido>/contenidos/tema[5]/subÍndice</contenido>
        </contenidoPanel>
         <destinoEnlaces panel="p1.3"/>
      </panel>

      <panel id= "p1.3" tam="400, 500">
        <contenidoPanel>
         <contenido>/contenidos/tema[1]/contenido</contenido>
         <contenido>/contenidos/tema[2]/contenido</contenido>
         <contenido>/contenidos/tema[3]/contenido</contenido>
         <contenido>/contenidos/tema[4]/contenido</contenido>
         <contenido>/contenidos/tema[5]/contenido</contenido>
        </contenidoPanel>
      </panel>
   </ventana>

</aplicacion>
```

La asignación de contenidos es un poco más tediosa por las razones anteriormente expuestas. Finalmente, la Figura 7.19 recoge el prototipo generado.

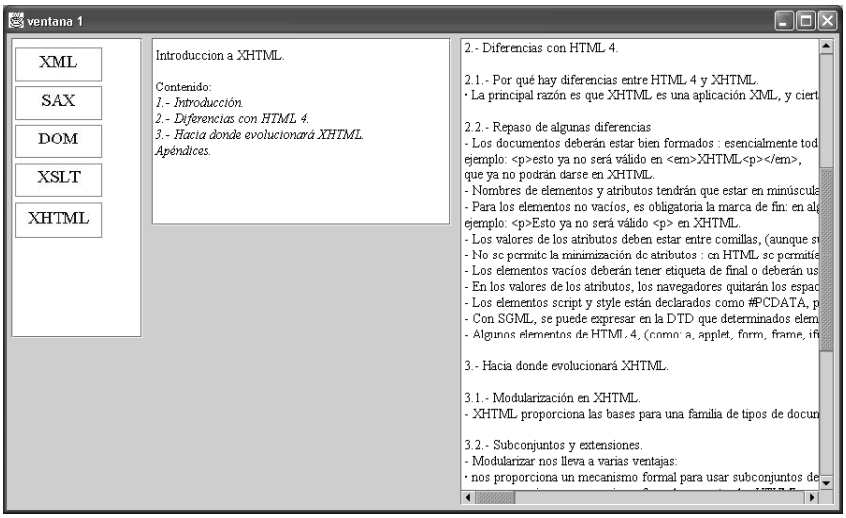

Figura 7.19 Prototipo GAP

7.3 e-Aula

7.3.1 Introducción

e-Aula (*http://147.96.25.235/clasevirtual/*) es un proyecto de aplicación de tecnologías XML para el *e-learning*. El objetivo principal del proyecto es crear un entorno de aula virtual que, mediante los lenguajes de marcado y las tecnologías web asociadas, especifique e implemente un sistema de e-learning de acuerdo al modelo de referencia de los estándares educativos. La metodología de creación del aula virtual pretende que el aula sea flexible, abierta y ampliable, de modo que se mejore la gestión, el acceso, la interactividad y la utilidad de la información educativa proporcionada mediante la red.
Para desarrollar este aula virtual es necesario desarrollar nuevas herramientas y entornos que permitan y simplifiquen:

a) La creación de contenidos educativos especialmente diseñados para la red (p.e. multimedia, hipermedia, interactivos) que permitan utilizar distintas estrategias educativas y
b) La gestión integrada de todas las actividades implicados en el proceso de aprendizaje (p.e. comunicaciones, publicación de cursos, relación con y entre los alumnos).

Este conjunto de aspectos educativos, de gestión y de creación de contenidos es lo que agrupamos bajo el denominador común de entorno de aula virtual o sistema de e-learning. Los dos pilares fundamentales para la creación de estos entornos son:

1) los estándares educativos en el nivel de diseño.
2) los lenguajes de marcado, destacando XML y las tecnologías web asociadas, en el nivel de implementación.

Aunque prácticamente todos los estándares tienen unos fundamentos comunes, existen discrepancias en sus modelos que incorporan particularidades. En este proyecto se comenzará utilizando el modelo ADL-SCORM [ADL-SCORM], que es el más estable actualmente, complementado y contrastado con las nuevas propuestas de IMS [IMS], aunque el objetivo es cumplir con las normas del estándar desarrollado por ISO, una vez que haya sido completado.

7.3.2 Análisis de requisitos y conceptualización

Los contenidos de la aplicación quedan caracterizados por el grafo de la Figura 7.20.

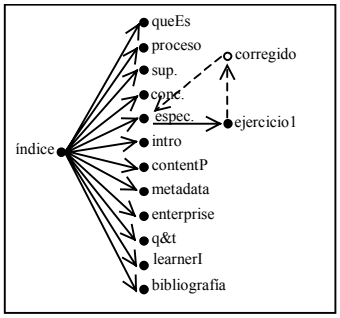

Figura 7.20 Grafo de contenidos de la aplicación.

Aunque el sistema incluye una serie de prerequisitos a la hora de acceder a los contenidos, así como tres niveles de adaptación al usuario, esta información trasciende la puramente hipermedia, y por tanto no tiene representación en el modelo. El esquema navegacional es el clásico HTML con un marco para el índice y otro para el resto de contenidos. La Figura 7.21 recoge este esquema navegacional.

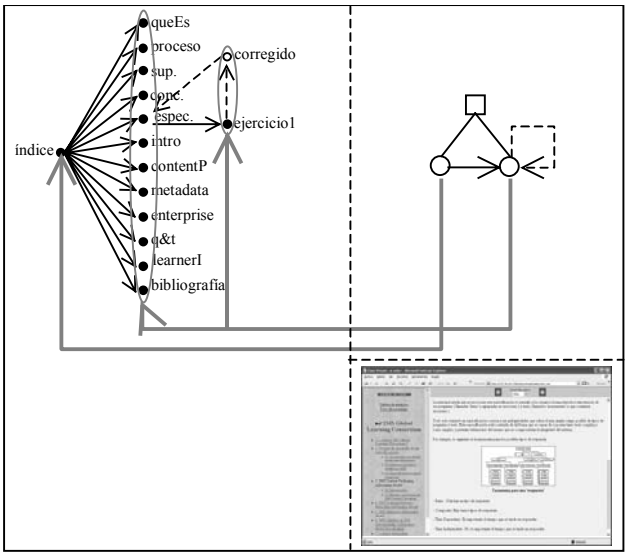

Figura 7.21 Esquema navegacional de la aplicación y relación con el grafo de contenidos.

7.3.3 Prototipado

En un ejercicio de ingeniería inversa no tiene demasiado sentido llevar a cabo esta fase. De todas formas veremos como es el prototipo para la el caso un par de contenidos.

La DTD de contenidos sería la siguiente.

Su instancia puede encontrarse en el Apéndice C. El documento de la aplicación que recoge el esquema navegacional de la Figura 7.21 es el siguiente.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="cursoIMS">

   <ventana id= "v1">
      <nombre>Ventana1</nombre>

      <panel id= "p1.1" tam="350, 300">
        <contenidoPanel>
         <contenidoDefecto>/curso/índice</contenidoDefecto>
        </contenidoPanel>
         <destinoEnlaces panel="p1.2"/>
      </panel>

      <panel id= "p1.2" tam="650, 700">
         <contenidoPanel>
         <contenidoDefecto>/curso/temas/tema[@id="intro"]</contenidoDefecto>
         <contenido>/curso/ejercicio</contenido>
         <contenido>/curso/corrección</contenido>
         <contenidoGrupo>/curso/temas</contenidoGrupo>
      </contenidoPanel>
         <destinoEnlaces panel="p1.2"/>
      </panel>
   </ventana>
```

```
</aplicacion>
```

Finalmente, la Figura 7.22 muestra el protipo GAP generado.

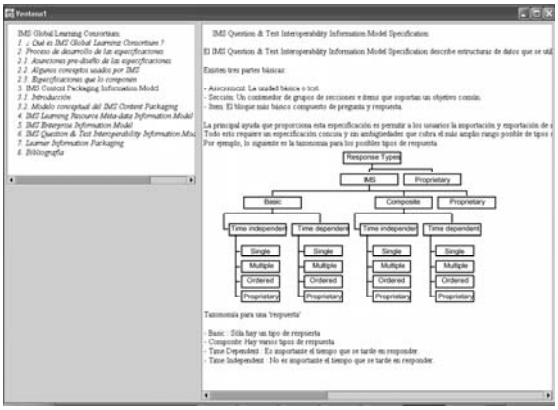

Figura 7.22 Prototipo GAP

7.4 Amazon

7.4.1 Introducción

Veamos ahora como modelar el sitio Web de Amazón (*http://www.amazon.com*) con el modelo Pipe. Nos centraremos exclusivamente en la parte generada dinámicamente. Debido a sus especiales características prestaremos especial atención a la semántica de navegación.

7.4.2 Análisis de requisitos y conceptualización

La idea de funcionamiento de Amazón es utilizar un formulario de búsqueda que genera una hoja de resultados. Esa hoja enlaza con descripciones de libros y con el carrito de compra. Cada descripción de libro permite al usuario adquirir dicho libro. A la hora de aplicar Pipe, vamos a suponer por simplicidad que la función g se comporta de manera determinista.

Es decir, el grafo de contenidos se asemeja al de la Figura 7.23. Nótese que salvo el formulario todos los contenidos se generan dinámicamente.

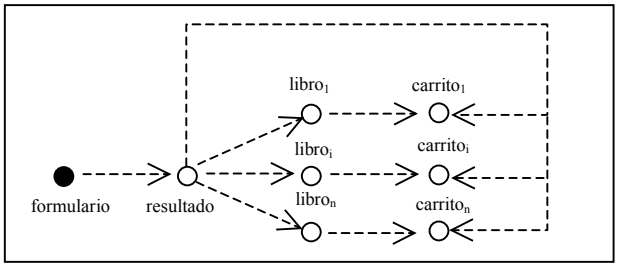

Figura 7.23 Grafo de contenidos

Si suponemos que C^0 = {formulario}, que CA^0 = {(formulario, botón)}, y $E^0 = \varnothing$, la primera aplicación de la función g para los contenidos de Amazón pude apreciarse en la Tabla 7.1 (supuesto que ha encontrado n libros).

k=1 g :	**CA0**	**C × A**
	(formulario, botón)	(resultado, total)

Tabla 7.1 Primera aplicación de la función g

Es decir, genera el resultado y las anclas a las descripciones de los libros y al carrito de compra. Por lo tanto tenemos:

C1 = {formulario} ∪ cg(formulario, botón) = {formulario} ∪ {resultado}
E1 = eg(formulario, botón) = {(formulario, botón, resultado, total)}
CA1 = ca(cg(formulario, botón)) = {(resultado, aL_1), (resultado, aCL_1), ..., (resultado, aL_i), (resultado, aCL_i), ..., (resultado, aL_n), (resultado, aCL_n) }

Si volvemos a aplicar la función g, obtenemos los resultados de la Tabla 7.2.

k=2 g:

CA1	C × A
(resultado, aL_1)	($libro_1$, total, {aCL_1}, {total})
(resultado, aCL_1)	($carrito_1$, total, ∅, {total})
.....	.....
(resultado, aL_i)	($libro_i$, total, {aCL_i}, {total})
(resultado, aCL_i)	($carrito_i$, total, ∅, {total})
.....	.....
(resultado, aL_n)	($libro_n$, total, {aCL_n}, {total})
(resultado, aCL_n)	($carrito_n$, total, ∅, {total})

Tabla 7.2 Segunda aplicación de la función g

Es decir, tenemos que,

C2 = C1 ∪ cg(CA1) = C1 ∪ { $libro_1$, $carrito_1$, ..., $libro_i$, $carrito_i$, $libro_n$, $carrito_n$ }
E2 = E1 ∪ eg(CA1) = E2 ∪ {(resultado, aL_1, $libro_1$, total), (resultado, aCL_1, $carrito_1$, total), ..., (resultado, aL_i, $libro_i$, total), (resultado, aCL_i, $carrito_i$, total), ..., (resultado, aL_n, $libro_n$, total), (resultado, aCL_n, $carrito_n$, total)) }
CA2 = ca(cg(CA1)) = { ($libro_1$, aCL_1), ..., ($libro_i$, aCL_i), ..., ($libro_n$, aCL_n) }

Finalmente si volvemos a aplicar la funcion g, obtenemos los resultados de la Tabla 7.3.

k=3 g:

CA2	C × A
(libro1, aCL_1)	($carrito_1$, total)
.....	.....
(libro1, aCL_i)	($carrito_i$, total)
.....	.....
(libron, aCL_n)	($carrito_n$, tota)

Tabla 7.3 Tercera aplicación de la función g

Es decir,

C3 = C2 ∪ {$carrito_1$, ..., $carrito_i$, ..., $carrito_n$ } = C2
E3 = E2 ∪ eg(CA2) = { ($libro_1$, aCL_1, $carrito_1$, total), ..., ($libro_i$, aCL_i, $carrito_i$, total), ..., ($libro_n$, aCL_n, $carrito_n$, total) }
CA3= ∅

El esquema navegacional de Amazon es relativamente sencillo. Hay una pantalla principal donde aparece el formulario de búsqueda. Tras esta hay otra pantalla donde aparece el resultado de la búsqueda. Desde esta pantalla se puede acceder al carrito de compra o a la pantalla que muestra la descripción de cada libro. Desde esta última pantalla también se puede acceder al carrito de compra. Es decir, el esquema navegacional queda recogido en la Figura 7.24. Nótese que en el sitio Web de Amazon la "vuelta atrás" está encomendada al navegador.

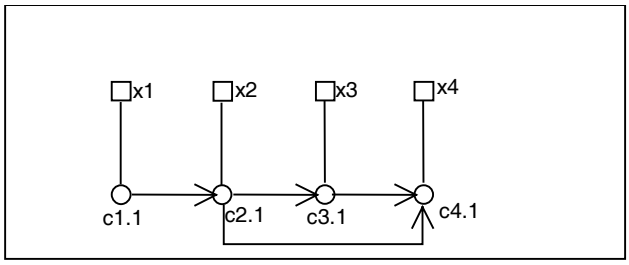

Figura 7.24 Esquema navegacional de Amazón

Veamos ahora cuales son las funciones d y l. Antes definámonos unos cuantos conjuntos,

Libros = { c | c ∈ C2, c = $libro_i$, i ∈ {1, ..., n}}
Carritos = { c | c ∈ C2, c = $carrito_i$, i ∈ {, ..., n}}
ResALib = { (o, ao, d, ad) | (o, ao, d, ad) ∈ E2, o = resultado, d = $libro_i$, i ∈ {1, ..., n}}
ResACar = { (o, ao, d, ad) | (o, ao, d, ad) ∈ E2, o = resultado, d = $carrito_i$, i ∈ {1, ..., n}}
LibACar = eg(CA2)

Nótese que esta partición de los conjuntos y los enlaces se corresponde con una partición de la función g en dos funciones, una de compra (g_c), y otra de descripción (g_l). De esta forma, tendríamos que:

Libros = $\prod_l g_l(ca(resultado)) = cg_l(ca(resultado))$
Carritos = $\prod_l g_c(ca(resultado)) = cg_c(ca(resultado))$

Nótese que también tenemos que,

Carritos = $\prod_l g_c(ca(Libros)) = cg_c(ca(Libros))$

De la misma forma,

ResALib = $eg_l(ca(resultado))$
ResACar = $eg_c(ca(resultado))$
LibACar = eg_c (CA2)

Con independencia de la aproximación elegida para obtener los conjuntos anteriores tenemos que,

d = { (**c1.1**, formulario, ∅), (**c2.1**, resultado, ∅), (**c3.1**, ∅, Libros), (**c4.1**, ∅, Carritos) }
l = { (**c1.1, c2.1, enlace**, eg(formulario, botón)), (**c2.1, c3.1**, enlace, ResALib), (**c2.1, c4.1, enlace**, ResACar), (**c3.1, c4.1, enlace**, LibACar) }

La Figura 7.25 ilustra a estos conjuntos.

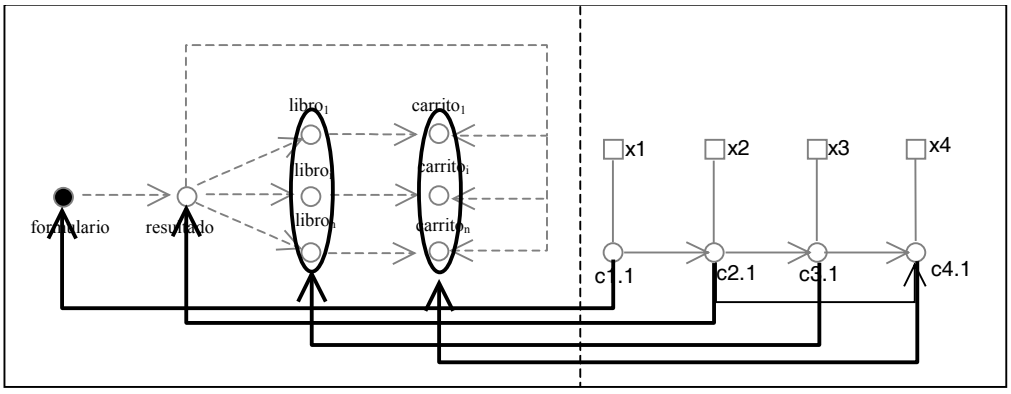

Figura 7.25 Relación entre el esquema navegacional y el grafo de contenidos. Las líneas en trazo grueso muestran la asignación de contenidos. A partir de esta puede deducirse la canalización de enlaces.

Veamos un ejemplo de que la semántica navegacional funciona como se desea.

- Empezamos activando la pantalla 1.

  a(x1).

  Activos1 = {x1, c1.1}
  Mostrar1= {(c1.1, formulario)}

  Resultado que puede apreciarse en la Figura 7.26.

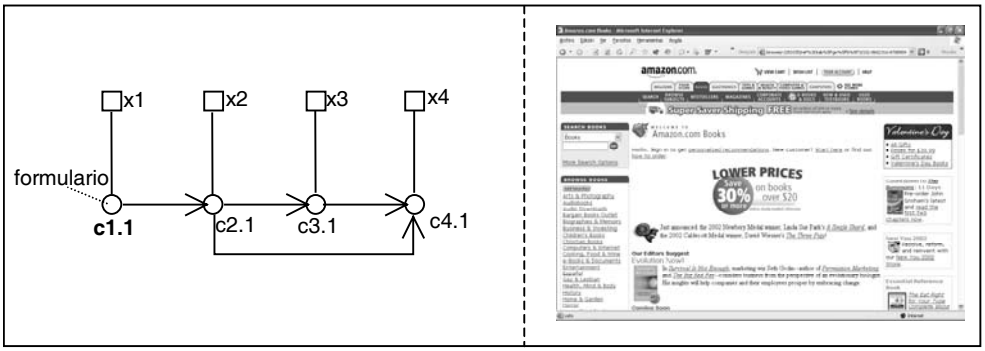

Figura 7.26 Activación de la primera pantalla

- Si introducimos una búsqueda y buscamos.

  f((formulario, botón), c1.1, Activos1, Mostrar1)

  $\exists 2.1((c1.1, c2.1, enlace) \in A \wedge eg(formulario, botón) \in l(c1.1, c2.1, enlace))$
  $\wedge (c1.1, formulario) \in Mostrar1$

  $\exists 1 \exists 2((x1, c1.1, conexión) \in A \wedge (x2, c2.1, conexión) \in A \wedge 1 \neq 2)$

  Activos2 = {x2, c2.1}
  Mostrar2 = {(c2.1, resultado)}

Resultado que puede apreciarse en la Figura 7.27.

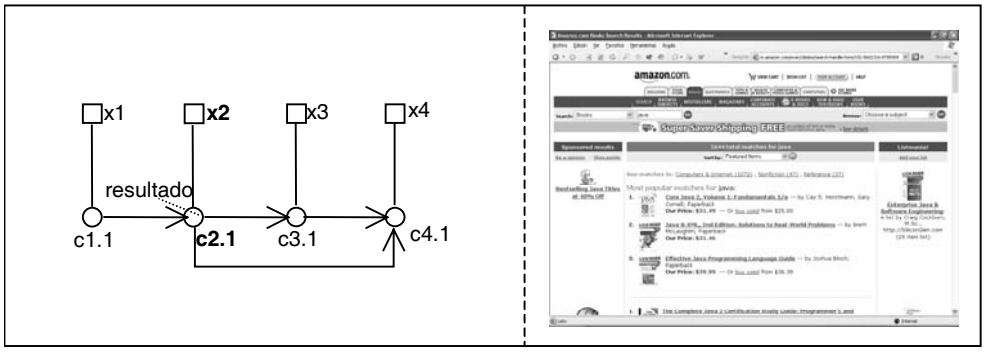

Figura 7.27 Búsqueda y generación de resultados

- Si ahora optamos por comprar el primer libro de la lista.

  f((resultado, aCL_1), c2.1, Activos2, Mostrar2)

  $\exists 4.1((c2.1, c4.1, enlace) \in A \wedge eg(resultado, aCL_1) \in l(c2.1, c4.1, enlace))$
  $\wedge (c2.1, resultado) \in Mostrar2$

  $\exists 2 \exists 4((x2, c2.1, conexión) \in A \wedge (x4, c4.1, conexión) \in A \wedge 2 \neq 4)$

Activos3 = {x4, c4.1}
Mostrar3 = {(c4.1, cg(resultado, aCL_1))} = {(c4.1, $carrito_1$)}

Nótese que eg(resultado, aCL_1) = (resultado, aCL_1, $carrito_1$, total) ∈ ResACar ∈ l(c2.1, c4.1, enlace). El resultado que puede apreciarse en la Figura 7.28.

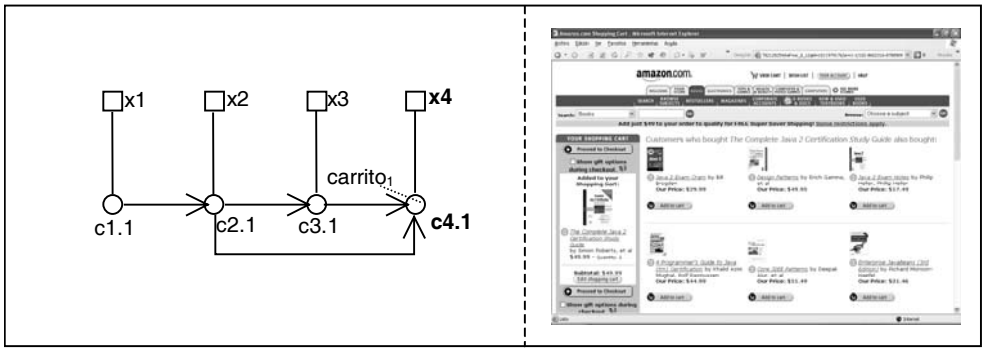

Figura 7.28 Compra del primer libro

- Si hubiésemos optado por ver primero la descripción del primer libro.

  f((resultado, aL_1), c2.1, Activos2, Mostrar2)

  ∃3.1((c2.1, c3.1, enlace) ∈ A ∧ eg(resultado, aL_1)∈ l(c2.1, c3.1, enlace))
  ∧ (c2.1, resultado)∈ Mostrar2

  ∃2∃3((x2, c2.1, conexión) ∈ A ∧ (x3, c3.1, conexión) ∈ A ∧ 2≠3)

  Activos3 = {x3, c3.1}
  Mostrar3 = {(c3.1, cg(resultado, aL_1))} = {(c3.1, $libro_1$)}

  Nótese que eg(resultado, aL_1) = (resultado, aL_1, $libro_1$, total) ∈ ResALib ∈ l(c2.1, c3.1, enlace). El resultado que puede apreciarse en la Figura 7.29.

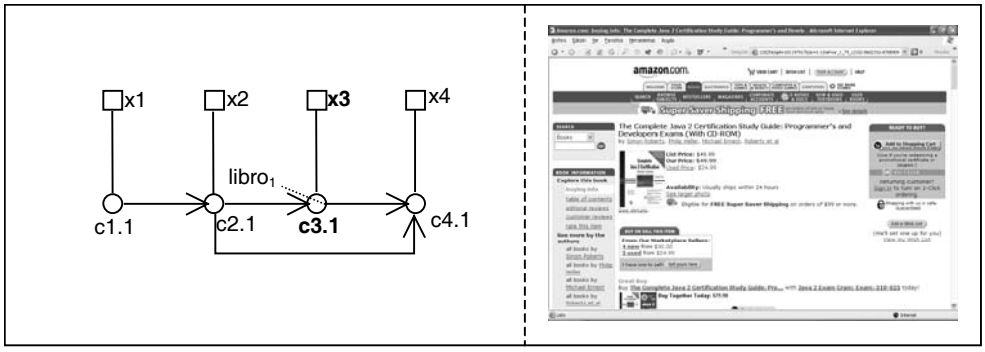

Figura 7.29 Visualización del libro

- Si ahora optamos por comprarlo.

  f(($libro_1$, aCL1), c3.1, Activos3, Mostrar3)

  ∃4.1((c3.1, c4.1, enlace) ∈ A ∧ eg($libro_1$, aCL_1)∈ l(c3.1, c4.1, enlace))
  ∧ (c3.1, $libro_1$)∈ Mostrar3

  ∃3∃4((x3, c3.1, conexión) ∈ A ∧ (x4, c4.1, conexión) ∈ A ∧ 3≠4)

Activos4 = {x4, c4.1}
Mostrar4 = {(c4.1, cg(libro1, aCL_1))} = {(c4.1, $carrito_1$)}

Nótese que eg($libro_1$, aCL_1) = ($libro_1$, aCL_1, $carrito_1$, total) ∈ LibACar ∈ l(c3.1, c4.1, enlace). El resultado puede apreciarse en la Figura 7.30.

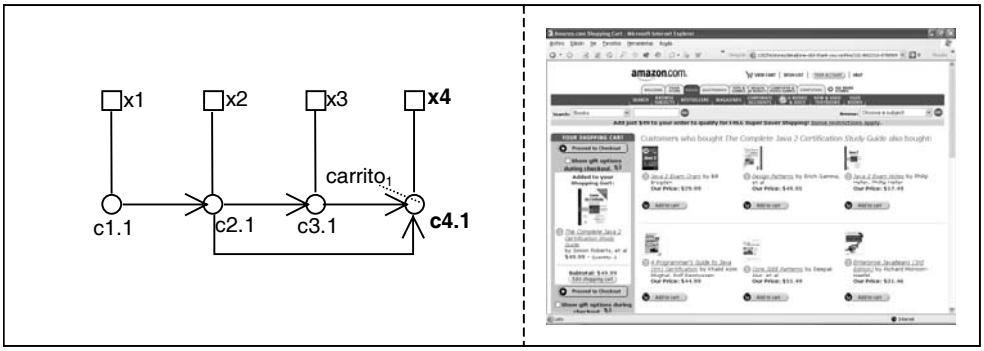

Figura 7.30 Compra del libro

- Si ahora optásemos por empezar una nueva búsqueda o una nueva selección (a través del navegador, ya que los contenidos no tienen enlaces "hacia atrás"), el proceso volvería a repetirse, y gracias a la naturaleza dinámica de los contenidos, obtendríamos el resultado deseado. No olvidemos que en última instancia la expresión $carritos_i$, representa un objeto dinámicamente generado perteneciente a $\prod_1 g_c(ca(resultado)) \cup \prod_1 g_c(ca(Libros))$.

7.4.3 Prototipado

En el caso de aplicaciones dinámicas el prototipado debe restringirse a una frase informativa. Queda como trabajo futuro la integración de la generación dinámica, aunque con mayor significado en metodología que en conceptualización.

La DTD de contenidos que simula el proceso dinámico de Amazon es la siguiente.

```
<!ELEMENT contenidos (formulario, resultado, libros, carritos)>

<!ELEMENT formulario (texto, botón)>
<!ELEMENT texto (#PCDATA)>
<!ELEMENT botón (#PCDATA)>
<!ATTLIST botón href IDREF #REQUIRED>

<!ELEMENT resultado (#PCDATA|enlace)*>
<!ATTLIST resultado id ID #REQUIRED>
<!ELEMENT enlace (#PCDATA)>
<!ATTLIST enlace href IDREF #REQUIRED>

<!ELEMENT libros (libro+)>
<!ELEMENT libro (#PCDATA|enlace)*>
<!ATTLIST libro id ID #REQUIRED>

<!ELEMENT carritos (carrito+)>
<!ELEMENT carrito (#PCDATA)>
<!ATTLIST carrito id ID #REQUIRED>
```

La instancia de dicha DTD puede encontrarse en el Apéndice C. El documento de la aplicación que captura el esquema navegacional de la Figura 7.24 (salvo la tubería de `c2.1` a `c4.1`) es el siguiente.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="amazon">
```

```
   <ventana id= "v1">
      <nombre>ventana 1</nombre>

      <panel id= "p1.1">
        <contenidoPanel>
         <contenidoDefecto>/contenidos/formulario</contenidoDefecto>
        </contenidoPanel>
      <destinoEnlaces panel="p2.1"/>

      </panel>
   </ventana>


   <ventana id= "v2">
      <nombre>ventana 2</nombre>

      <panel id= "p2.1">
        <contenidoPanel>
         <contenido>/contenidos/resultado</contenido>
        </contenidoPanel>
         <destinoEnlaces panel="p3.1"/>
      </panel>
   </ventana>


   <ventana id= "v3">
      <nombre>ventana 3</nombre>

      <panel id= "p3.1">
        <contenidoPanel>
         <contenidoGrupo>/contenidos/libros</contenidoGrupo>
        </contenidoPanel>
        <destinoEnlaces panel="p4.1"/>
      </panel>
    </ventana>

   <ventana id= "v4">
      <nombre>ventana 4</nombre>

      <panel id= "p4.1">
        <contenidoPanel>
         <contenidoGrupo>/contenidos/carritos</contenidoGrupo>
        </contenidoPanel>
      </panel>
    </ventana>

</aplicacion>
```

Finalmente la Figura 7.31 captura las ventanas GAP generadas.

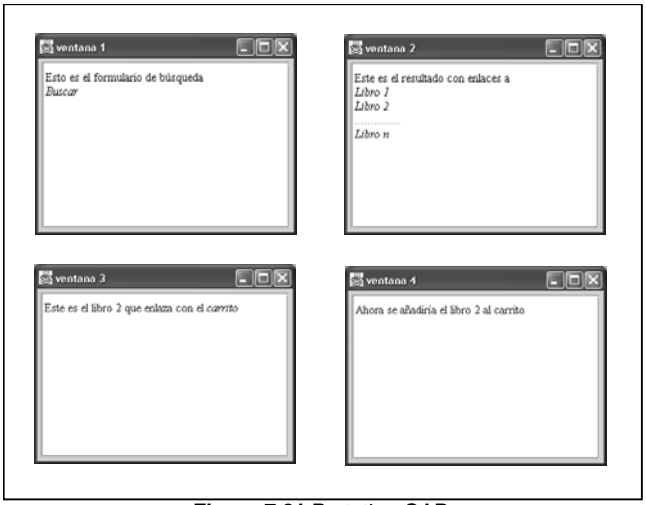

Figura 7.31 Prototipo GAP

7.5 Conclusiones

Como hemos podido ver a lo largo de este capítulo, Pipe sirve como herramienta de conceptualización de diverso tipo de aplicaciones hipermedia. Entre estas se encuentran las diseñadas para su uso en CD-ROM, o para su uso en Internet. Además es capaz de caracterizar tanto aplicaciones eminentemente estáticas, como las eminentemente dinámicas. Por otro lado los diagramas Pipe pueden ser utilizados como mapas conceptuales de sitios, Web, muy necesarios para situar al lector en un contexto de acceso hipermedia.

También hemos comprobado como utilizando las descripciones XML del modelo Pipe, y el generador automático de prototipos (el cual utiliza la semántica de presentación por defecto del modelo hipermedia) se pueden construir de manera automática prototipos. Estos prototipos son de gran utilidad para determinar la componente exclusivamente hipermedia de la aplicación, con independencia de técnicas concretas de diseño y/o implementación. Además, gracias a las ventajas que XML proporciona como formato de intercambio, la información utilizada para generar los prototipos puede reutilizarse después en la fase de construcción de la aplicación.

Por último, si observamos los diagramas Pipe vemos como las herramientas diseñadas para su uso en CD-ROM tienen un esquema navegacional (o interfaz de usuario) bastante más complejas que las diseñadas para su despliegue en Internet. La razón de este hecho puede deberse a las limitaciones impuestas por HTML, que en la práctica solamente pueden salvarse utilizando lenguajes de *script* (como JavaScript, por ejemplo). También hemos visto como la parte dinámica de las aplicaciones, así como las aplicaciones eminentemente dinámicas (como Amazón) tienden a utilizar esquemas navegacionales bastante sencillos.

8 Conclusiones y trabajo futuro

8.1 Conclusiones

Son varias las conclusiones que hemos extraído durante la realización de esta tesis, muchas de ellas provenientes del estudio sobre el estado del arte. Así, a la hora de caracterizar el diseño de una aplicación hipermedia disponemos de multitud de sistemas de representación de muy diversa naturaleza y origen. Aunque hemos hecho una clasificación razonada en el Capítulo 2 mostrando sus principales ventajas e inconvenientes, vamos a indicar en este apartado las conclusiones personales de cada uno de ellos. La valía del modelo Dexter radica en la claridad de ideas que impone al dominio hipermedia. En particular, su separación en tres niveles es, una de las mayores aportaciones en el área. Por el contrario su alto nivel de abstracción dificulta su uso en práctica. El modelo Amsterdam expresa bastante bien las relaciones temporales y de sincronización, y su representacíon a través del lenguaje SMIL le hace idóneo para caracterizar aplicaciones con un gran número de restricciones temporales. En su contra juega la representación del contexto a través de la noción de componente compuesto, lo cual, desde nuestro punto de vista, obstaculiza su aplicabilidad. La máquina abstracta de hipertexto es demasiado limitada para representar aplicaciones actuales, pero su simplicidad y precisión la hacen una valiosa herramienta para caracterizar los contenidos hipermedia y sus relaciones (de hecho, el grafo de contenidos Pipe no es ni más ni menos que una representación HAM). El modelo de hipergrafos, es a nuestro juicio, el de menor nivel, ya que en la práctica debe recurrir a primitivas para caracterizar la navegación por el hipergrafo. En contraposición el modelo Trellis es uno de los más elegantes, ya que descansa en la estructura de red de Petri para representar las aplicaciones hipermedia. La parte negativa viene por sus limitaciones a la hora de caracterizar las aplicaciones actuales. HDM es un modelo que a nuestro entender complica sobremanera el uso de tecnologías relacionales en el dominio hipermedia. En particular el paso de enlaces abstractos a concretos puede ser un buen ejemplo de dichas complicaciones. En cambio, RMDM sí que representa un modelo conceptual bastante más claro para integrar el modelo relacional en el contexto hipermedia. Sin embargo, el esquema navegacional que permite describir es mínimo (cercano al de HAM). Labyrinth representa una evolución en los sistemas de representación, ya que permite caracterizar cualquier evento computacional en el contexto de una aplicación hipermedia. El precio de esta potencia es la necesidad de incluir código (o pseudocódigo) en el modelo para representar, precisamente, la respuesta a estos eventos. Por último OOHDM es muy potente al ser una metodología de diseño. El inconveniente es el gran número de diagramas que involucra en la representación de las aplicaciones, y la brecha que existe entre estas y la aplicación hipermedia. El modelo hipermedia propuesto en esta tesis intentará eliminar los mayores inconvenientes de los sistemas de representación hipermedia analizados, manteniendo sus ventajas.

Pensamos que la ingeniería hipermedia es una de las áreas que más investigación necesita dentro del dominio hipermedia, y precisamente esta memoria representa otro esfuerzo en este sentido. No hay un gran número de modelos de proceso, ya que simplemente se han adaptado los de mayor éxito en el área de la informática (los evolutivos) al campo hipermedia. En contraposición existen multitud de metodologías. Nosotros nos hemos centrado en estudiar aquellas que a nuestro entender tienen una caracterización conceptual más clara: las basadas en sistemas de representación hipermedia. En este sentido, los comentarios sobre las metodogías son análogos a los de los modelos en los que se basan. Por último, la aproximación DTC es otra forma de construir aplicaciones software genéricas que presenta importantes ventajas en el apartado de mantenibilidad. Además en el seno de nuestro grupo la hemos aplicado con éxito en el ámbito hipermedia.

Respecto a los lenguajes de marcado debemos resaltar el éxito de XML y tecnologías relacionadas en todo el área de la informática, y no solamente en el dominio hipermedia. A nuestro juicio, este éxito está sobrevalorado, y pensamos que con el tiempo las tecnologías de marcado encontrarán su verdadero nivel de aplicabilidad real. Una constante en este área ha sido el poco éxito de lenguajes complejos frente a variantes menos potentes, pero más sencillas. Por ejemplo, SGML, lenguaje restringido al ámbito editorial, ha sido sustituido por XML, extendido en todas las áreas de la informática. HyTime jamás tuvo éxito, en contraposición a aproximaciónes más sencillas como HTML. También tenemos en este área el mismo dilema que presentaban los sistemas de representación hipermedia. Ser capaces de responder a eventos genéricos (UIML, por ejemplo), o restringir el número de eventos a los que puede responder el sistema que interpreta al lenguaje (SMIL, por ejemplo). En el primer caso tendremos más potencia, pero mayor complejidad. En el segundo caso tendremos todo lo contrario. Aplicando la heurística sobre la complejidad los primeros parecen estar abocados al fracaso. No queremos en esta memoria lanzar ninguna predicción. Simplemente constatamos una sensación que tenemos.

Pero, tal y como cabría esperar, no solo hemos extraído conclusiones del estado del arte. Veamos a continuación cuales son las principales conclusiones que hemos extraído del trabajo que representa la mayor aportación de esta tesis: Pipe, Plumbing y PlumbingXJ.

Han pasado ya muchos años desde mi primer contacto con el mundo de la hipermedia, y bastante más tiempo desde que empecé a programar en el lenguaje BASIC de mi Atari 600XL. Desde este primer lenguaje he aprendido distintos lenguajes informáticos (de programación o de otra naturaleza): Pascal, C, C++, Prolog, Haskell, SQL, SGML, XML, HTML... También me he visto envuelto en diversos proyectos y experiencias, bien como alumno, bien en la empresa, o bien como profesor en la universidad. La mayor conclusión extraída de este corto pero interesante viaje es que los programadores, como equipo o como individuo, tienen una tendencia natural a lo que podríamos llamar "programación a corto plazo". Esta consiste en proporcionar una solución correcta y completa, pero casi siempre con dos grandes limitaciones. Primera, el programador siempre elegirá el lenguaje (y paradigma) con el que se sienta más cómodo. Segunda, siempre se primará la fecha de entrega frente a la mantenibilidad del código. Poco podemos hacer frente a la primera cuestión, salvo proporcionar a los programadores una vasta formación en los diversos paradigmas y lenguajes para que elijan en cada ocasión aquel que mejor se adecue a sus necesidades. Frente a la segunda característica debemos introducir una fuerte política de ingeniería del software tanto en el nivel de gestión, como en el de desarrollo. En la parte de gestión contamos con una valiosa arma: el plan de proyecto del software. En la parte de diseño disponemos de los formalismos de análisis y diseño, apoyados tanto en notaciones estándar como en patrones de diseño.

En esta tesis hemos analizado excelentes sistemas de representación cuya finalidad es caracterizar el diseño de las aplicaciones hipermedia. Todos estos sistemas presentan importantes ventajas, y el trabajo desarrollado en esta tesis no viene a sustituir esta panoplia de herramientas con las que enfrentarnos al desarrollo de aplicaciones hipermedia (viene a mi memoria la portada del excelente libro de compiladores de Aho, Setti y Ullman), sino a integrarse con el fin de aumentar las probabilidades de éxito de los desarrolladores. Así el modelo Pipe no solo cumple los mínimos exigibles propuestos por modelos como el Dexter o el Amsterdam, sino que además presenta interesantes ventajas. La primera es su alto nivel de abstracción basado exclusivamente en de grafos y funciones de relación y navegación. Dicha abstracción le permite englobar y extender, por ejemplo, a los modelos hipermedia de corte relacional. La segunda ventaja es su capacidad para representar aplicaciones hipermedia dinámicas, cuyo uso se está extendiendo como la pólvora en Internet. Finalmente proporciona una semántica de navegación por defecto, permitiendo con las tecnologías adecuadas, generar aplicaciones hipermedia basadas en las estructuras Pipe.

Las ventajas que acabamos de enumerar ponen de manifiesto la potencia expresiva de Pipe, pero una de sus mayores ventajas es el uso que se hace de él. A diferencia de los sistemas de representación clásicos concebidos como herramientas de diseño sobre las cuales basar la implementación, Pipe está concebido como una herramienta de conceptualización. ¿Es esta una sutil diferenciación o una mera cuestión de marketing? Salvo Dexter, tan genérico que ni siquiera compromete una notación, todos los sistemas de representación hipermedia considerados están ideados como notación de diseño sobre la cual implementar. En efecto, las estructuras Amsterdam son tan específicas que su implementación natural es un lenguaje diseñado sobre ellas, tal y como lo es SMIL. La máquina abstracta de hipertexto quizás pudiera ser utilizada como herramienta de conceptualización (de hecho el grafo de contenidos Pipe está inspirado en ella), pero sus limitadas capacidades la excluyen de su uso para caracterizar las modernas aplicaciones hipermedia. El modelo Trellis basado en redes de Petri sufre el mismo problema que el Amsterdam, siendo casi exclusivamente viable una implementación basada en el sistema αTrellis. El sistema de hipergrafos de Tompa, tal y como vimos, solamente funciona si se dispone de un sistema de bases de datos capaz de soportar sus primitivas de acceso. Los modelos relacionales, HDM y RMDM, por definición restringen su área de aplicación al de los sistemas de bases de datos relacionales. La metodología OOHDM no tiene sentido fuera del contexto orientado a objetos, bastante distante de otras aproximaciones como la propuesta por HTML. Finalmente Labyrinth es capaz de responder a cualquier evento, pagando el precio de tener que proporcionar tanto el código como las estructuras de datos necesarias para llevar a cabo esta función. Por tanto, en la práctica, el mayor inconveniente de estos sistemas de representación es que obligan al programador a utilizar unas técnicas de implementación basadas casi exclusivamente en la potencia expresiva del modelo. Como resultado los programadores terminan por no utilizar estos sistemas de representación hipermedia.

Podría desprenderse del párrafo anterior la falta de convencimiento en el uso de modelos hipermedia, pero nada más lejano del ánimo del autor de este texto. Si tenemos una base de datos relacional y queremos

proporcionar acceso hipermedia de manera sencilla y elegante, nada mejor que utilizar el modelo RMDM. Pero ¿qué hacer en el caso de aquellas aplicaciones hipermedia en las que no tengamos claro ni siquiera que tipo de tecnología utilizar en la implementación? No olvidemos que las aplicaciones hipermedia son aplicaciones software, y sufren por tanto los mismos problemas que estas a la hora de acometer su construcción. La mayor parte de estos problemas pueden solventarse con una especificación correcta, completa, y estable de las principales características de las aplicaciones. En el caso de aplicaciones hipermedia contar con esta especificación es una tarea más compleja si cabe, ya que los clientes son los encargados de proporcionar los contenidos y el esquema navegacional de los mismos. Parece que hemos llegado a un bucle del cual no podemos salir. Para construir tenemos que diseñar. Para diseñar tenemos que especificar. Para especificar tener claro que es lo que hace la aplicación. Para esto debemos construir... Hemos descubierto la necesidad de un modelo de proceso iterativo-incremental, basado en prototipos cuyo diseño no comprometa en exceso la implementación. Necesitamos por tanto el modelo de proceso de Fraternali, el cual incluye una fase de conceptualización previa al diseño y construcción que sirve para identificar los elementos fundamentales de la aplicación hipermedia. Son precisamente las ventajas Pipe las que le hacen idóneo para integrarse en el modelo de proceso de Fraternali.

Al insertar Pipe en el modelo de proceso de Fraternali, obtenemos el modelo de proceso Plumbing, el cual se beneficia de las propiedades Pipe, y de las propias características proporcionadas por el modelo de proceso original. En particular, el mayor número de iteraciones en la construcción de las aplicaciones se producen en las fases de conceptualización y prototipado, menos costosas que diseño y desarrollo. Además estas iteraciones previas permiten dar un mayor protagonismo al cliente, que se mantiene ajeno a características concretas de implementación. Finalmente, Plumbing es lo suficientemente abstracto como para encontrarse abierto a diferentes concreciones. En particular PlumbingXJ es una de ellas.

PlumbingXJ utiliza documentos XML para proporcionar soporte a las representaciones Pipe y un programa Java para implementar su semántica de navegación por defecto, y dar cabida a los procesos subordinados. La primera cuestión sería discutir si PlumbingXJ es modelo de proceso o metodología. Entrar en esta discusión sería similar a cuestionar si el modelo de proceso unificado, basado en UML, y porque no decirlo en el uso de herramientas CASE como Rational Rose, es modelo de proceso o metodología. Nosotros preferimos denominarlo modelo de proceso, ya que su fin último es no comprometer ni el diseño ni la implementación final.

La primera ventaja de PlumbingXJ es probar la viabilidad de Plumbing. Además, el uso de XML y Java tiene la innegable ventaja de independencia de la plataforma, haciendo portables los prototipos GAP a entornos heterogéneos. Pero el uso de XML también presenta otras ventajas. Permite estructurar el dominio de los contenidos. Es un formato de intercambio electrónico universal. Su legibilidad permite una mejor interacción entre el equipo de desarrolladores y el cliente. Esto redunda en una mayor implicación del cliente a la hora de construir la aplicación. El documento de la aplicación representa un texto de mayor declaratividad que el código fuente, permitiendo un desarrollo rápido de aplicaciones por parte de los programadores, alternativo al uso exclusivo de herramientas CASE. Pero no olvidemos las ventajas que proporciona la conceptualización Pipe en la fase de prototipado. El documento de la aplicación no es más que una concreción de las estructuras Pipe proporcionadas en la fase anterior. Por tanto la construcción del mismo está totalmente guiada por el esfuerzo previo. La independencia total entre el documento de contenidos y de la aplicación permite hacer cambios ortogonales en los contenidos y en la navegación de los mismos. Por último, la semántica de navegación por defecto Pipe permite construir prototipos de manera automática basados en los documentos anteriores. Además, en último término, toda la información generada en las fases de conceptualización y prototipado no comprometen la implementación final, ya que la representación de la información en formato XML permite su exportación a casi cualquier formato electrónico.

8.2 Trabajo futuro

A lo largo de esta tesis hemos ido planteando cuestiones y problemas que no hemos resuelto. Este apartado hace enumeración de dichas reflexiones que han sido postergadas hasta la presentación de esta memoria.

Respecto a Pipe necesitamos estudiar, si tal y como hemos planteado en esta tesis, es necesario considerar generaciones dinámicas del esquema navegacional, o por el contrario es suficiente la caracterización actual. Nos inclinamos a pensar que no va a ser necesaria esta ampliación, pero dejamos la puerta abierta. También

sería necesario hacer un estudio en profundidad de la aproximación seguida en el desarrollo de semánticas denotacionales, para contrastarlas con las representaciones Pipe.

Una cuestión enclavada en trabajo futuro, pero que la realidad la sitúa en trabajo actual, es la ampliación de GAP hasta soportar todas las capacidades expresivas proporcionadas por Pipe. De hecho, en la actualidad, GAP está más cercano a un prototipo de lo que debería ser, que a la propia versión final. Como ya habíamos demostrado la viabilidad de las técnicas PlumbingXJ para representar a Pipe optamos por congelar el desarrollo GAP hasta la presentación de esta tesis, pero en la agenda de nuestro grupo de investigación la finalización de GAP ocupa un puesto preferente. En particular ahora estamos centrados en la incorporación de diversas técnicas que permitan la generación de aplicaciones con enlaces y contenidos dinámicamente generados.

Otra cuestión que ha surgido a lo largo de nuestra investigación es la búsqueda de otras concreciones de Plumbing. En particular nos parece especialmente interesante la representación del grafo de contenidos como un diagrama entidad relación (de manera similar a como a partir de un diagrama RMDM se obtenía una representación HAM de la aplicación). Esto conllevaría la sustitución de las expresiones XPath por expresiones SQL que actuarían sobre los contenidos. Debemos estudiar esta posibilidad, tanto en su versión estática como dinámica. Con independencia de los resultados finales, ya tenemos nombre para la aproximación (por algún sitio debíamos empezar): *PlumbingSQLJ*.

Somos conscientes de que el uso a gran escala del formalismo propuesto por Pipe puede resultar bastante poco atractivo. De hecho, una de las mayores críticas a los sistemas de representación hipermedia era su falta de soporte CASE. No deseamos incluir en el mismo error y por tanto vemos prioritario el desarrollo de PlumbingMatic. Esta herramienta no solo facilitaría la conceptualización de las aplicaciones hipermedia, sino que su integración junto a GAP permitiría cubrir totalmente el primer ciclo del modelo de proceso de Fraternali. En cualquier caso, el desarrollo de esta herramienta CASE estará totalmente guiado por la formalización proporcionada por el modelo Pipe.

La disponibilidad de PlumbingMatic facilitaría sobremanera la aplicación de PlumbingXJ al desarrollo de distintas aplicaciones hipermedia. En esta tesis hemos optado por un ejercicio de ingeniería inversa debido al relativo alto tiempo de desarrollo de este tipo de aplicaciones. En el futuro, y con más calma, necesitamos, y debemos aplicar PlumbingXJ en un ejercicio de *ingeniería directa*.

Uno de los principales resultados que esperamos obtener de la aplicación de PlumbingXJ es la confirmación de si debemos aplicar las técnicas de conceptualización y prototipado a las fases de diseño y desarrollo. De esta forma obtendríamos una metodología concreta de desarrollo. Otra posibilidad sería utilizar desarrollos basados en componentes software reutilizables que permitiesen un desarrollo flexible de aplicaciones a partir de las descripciones Pipe. Precisamente, los procesos subordinados representa el embrión de estas técnicas.

9 Referencias

[Abrams 00] Abrams, M. *Device-Independent Authoring with UIML* W3C Workshop on Web Device Independent Authoring. HP Labs, Bristol, 3-4 octubre, 2000 .

[ADL-SCORM] *ADL/SCORM*. http://adlnet.org/SCORM/scorm_index1.cfm, 2001.

[Aho 90] Aho, A.V, Sethi, R.S, Ullman, J.D. *Compiladores: principios, técnicas y herramientas*. Addison-Wesley Iberoamericana, 1990.

[Balasubramanian 97] Balasubramanian, V., Bashian A., Porcher, D. *A Large-scale Hypermedia Application Using Document Management and Web Technologies*. The Eight ACM International Hypertext Conference, Hypertext 97. Southhampton, 6-11 abril, 1997.

[Barry 01] Barry, C., Lang, M. *A Survey of Multimedia and Web Development Techniques and Methodology Usage*. IEEE Multimedia (Special Issue on Web Engineering) 8 (3), 52-60, 2001.

[Baumeister 99] Baumeister, H., Koch, N., Mandel, L. *Towards a UML Extension for Hypermedia Design*. The Second International Conference on The Unified Modeling Language, UML'99. Fort Collins, 28-30 octubre, 1999.

[Berners-Lee 01] Berners-Lee, T. *HTML Overview*. http://www.zdnet.com/devhead/resources/tag_library/history/html.html, 2001.

[Bodner 00] Bodner, R., Chignell, M. *Dynamic Hypertext: Querying and Linking*. ACM Computing Surveys, 31 (4), 1999.

[Booch 96] Booch G. *Análisis y diseño orientado a objetos. Segunda edición*. Addison-Wesley/Díaz de Santos, 1996.

[Booch 99] Booch G., Rumbaugh J., Jacobson I. *El lenguaje unificado de modelado*. Addison-Wesley, 1999.

[Bourret 01] Bourret, R. *Mapping DTDs to Databases*. http://www.xml.com/lpt/a/2001/05/09/dtdtobdbs.html, 2001.

[Bryan 98] Bryan, M. *Guidelines for using XML for Electronic Data Interchange*. http://www.xmledi-group.rog/xmledigroup/guide.htm, 1998.

[Burnard 97] Burnard, L. *SGML on the Web: Too Little Too Soon, or Too Much Too Late*. Computer and Texts 15 (agosto), 1997.

[Bush 45] Bush, V. *As We May Think*. The Atlantic Monthly, 176 (July) 101-108, 1945.

[Campbell 88] Campbell B., Goodman J.M. *HAM: A general purpose hypertext abstract machine*. Communications of the ACM 31(7), 856-861, 1988.

[Chen 76] Chen, P. *The entity-relationship approach: toward a unified view of data*. ACM Transactions on Database Systems 1 (1), 1976.

[Conklin 87] Conklin, J. *Hypertext: An Introduction and Survey*. ACM Computing Surveys 31 (3), 17-41, 1987.

[Davis 92] Davis, H., Hall, W., Heath, I., Hill, G. *Towards An Integrated Information Environment With Open Hypermedia Systems*. ACM European Conference on Hypertext, ECHT 92. Milán, 30 noviembre – 4 diciembre, 1992.

[De Bra 00] De Bra, P., Brusilovsky, P., Geert-Jan, H. *Adaptive Hypermedia: From Systems to Framework*. ACM Computing Surveys, 31 (4), 2000.

[Deakin 01] Deakin, N. *XUL Tututorial*. http://www.xulplanet.com/tutorials/xultu/, 2001.

[DeRose 94] DeRose S.J., Durand D.G. *Making Hypermedia Work. A User´s Guide to HyTime*. Kluwer Academic Publishers, 1994

[Díaz 94] Díaz P., Aedo I., Panetsos F. *Labyrinth, an abstract model for hypermedia applications. Description of its static components*. Information Systems 19 (4), 33-45, 1994.

[Díaz 97] Díaz P., Aedo I., Plaza A. *Modelos para el diseño de sistemas hipermediales*. Informática y Automática, 30 (2), 51-61, 1997.

[Díaz 99] Díaz, P., Aedo., I, Panetsos, F. *A Methodological Framework for the Conceptual Design of Hypermedia Systems*. Hypertexts and Hypermedia: Products, Tools, Methods, H2PTM 99. París, 23-24 septiembre, 1999.

[Díaz 01] Díaz P., Aedo I., Panetsos F. *Modelling the dynamic behaviour of hypermedia applications*. IEEE Transactions on Software Engineering, 27 (6), 550-572, 2001.

[DOC 96] *Dictionary of Computing. Cuarta edición*. Oxford University Press, 1996.

[Dodds 01a] Dodds, L. *Does XML Query Reinvent the Wheel?* http://www.xml.com/lpt/a/2001/02/28/deviant.html, 2001.

[Dodds 01b] Dodds, L. *XPointer and the Patent*. http://www.xml.com/pub/a/2001/01/17/xpointer.html, 2001.

[ECS 00] *Encyclopedia of Computer Science. Cuarta edición*. A. Ralston, E.D. Reilly y D. Hemmendinger (eds.). Nature Publishing Group, 2000.

[Elmasri 97] Elmasri R., Navathe S.B. *Sistemas de bases de datos: conceptos fundamentales*. Addison-Wesley, 1997.

[EMLa] *Educational Modelling Language*. http://eml.ou.nl/eml/, 2001.

[EMLb] *Election Markup Language*. http://xml.coverpages.org/eml.html

[Fernández 97] Fernández, B., Fernández-Valmayor A., Navarro, A. *Extending Web educational applications via SGML structuring and content-based capabilities*. F. Verdejo y G. Davies (eds.), The Virtual Campus: Trends for Higher Education and Training, pp 244-259. Chapman & Hall, Londres, 1997.

[Fernández 98a] Fernández, B., Cigarrán, J., Navarro, A., Fernández-Valmayor, A. *Using Automatic Methods for Structuring Conceptual Knowledge in Intelligent Learning Environments*. Intelligent Tutoring Systems, 4th International Conference, ITS98, pp 264-273. San Antonio, 16–19 agosto, 1998. Publicado como Lecture Notes in Computer Science (Vol. 1452), Springer-Verlag, Berlin, 1998.

[Fernández 98b] Fernández, B., Navarro, A., Cigarrán, J., Fernández-Valmayor, A. *Declarative Mark-Up Languages as a Tool for Developing Educational Hypermedias*. Intelligent Tutoring Systems, 4th International Conference, ITS98, pp. 607. San Antonio 16–19 agosto, 1998. Publicado como Lecture Notes in Computer Science (Vol. 1452), Springer-Verlag, Berlin, 1998.

[Fernández 98c] Fernández, B., Cigarrán, J., Navarro, A., Fernández-Valmayor, A. *Applying Formal Concept Analysis to Domain Modeling in an Intelligent Help System*. XV IFIP World Computer Congress, IT & KNOWS 98, pp 387-399. Viena y Budapest, 31 agosto – 4 septiembre, 1998.

[Fernández 98d] Fernández, B., Navarro A., Cigarrán J., Fernández-Valmayor A. *Using Standard Markup in the design and development of Web educational software*. XV IFIP World Computer Congress, Teleteaching 98, pp 303-312. Viena y Budapest, 31 agosto – 4 septiembre, 1998.

[Fernández 98e] Fernández, B., Navarro A., Cigarran J.M., Fernandez-Valmayor A. *Integration of Formal Concept Analysis in a Knowledge-Based Assistant*. 11th International Conference on Industrial and Engineering Applications of artificial Intelligence and Expert Systems, IEA-98-AIE, pp 112-123. Benicasim, 1–4 junio 1998. Publicado como Lectures Notes in Artificial Intelligence (Vol. 1415), Springer-Verlag, Berlin, 1998.

[Férnandez-Valmayor 99] Fernández-Valmayor, A., López-Alonso C., Fernández, B., Sere, A. *Integrating an Interactive Learning Paradigm for Foreign Language Text Comprehension into a Flexible Hypermedia System*. International Working Conference on Building University Electronic Educational Environments. IFIP WG3.2 and WG 3.6 joint conference. California, 4-6 agosto, 1999.

[Fraternali 99] Fraternali, P. *Tools and Approaches for Developing Data-Intensive Web Applications: A Survey*. ACM Computing Surveys 31 (3) , 227-263, 1999.

[Fuchs 97] Fuchs, M. *Domain Specific Languages for ad hoc distributed applications*. First Conference on Domain Specific Languages. Sta. Barbara, 15-17 Octubre, 1997.

[Garg 88] Garg P.K. *Abstraction mechanisms in hypertext*. Communications of the ACM 31 (7), 862-870, 1988.

[Garzotto 93] Garzotto F., Paolini P., Schwabe D. *HDM: A model-based approach to hypertext application design*. ACM Transactions on Information Systems 11(1), 1-26, 1993.

[Garzotto 95] Garzotto F., Mainetti L., Paolini P. *Hypermedia design, analysis, and evaluation issues*. Communications of the ACM 38 (8), 74-86, 1995.

[Ginige 97] Ginige, A., Lowe, D. *Hypermedia Engineering: Process for developing large hypermedia systems*. Tutorial at The Eighth ACM Conference on Hypertext. Southampton, 9-11 abril 1997.

[Goldfarb 90] Goldfarb C.F. *The SGML Handbook*. Oxford University Press, 1990.

[Göschka 99] Göschka, K. M., Falb, J. *Dynamic Hyperlink Generation for Navigation in Relational Databases*. The 10th ACM Conference on Hypertext and Hypermedia, Hypertext 99, pp 23-24. Darmstadt, 21-28 febrero, 1999.

[Halasz 94] Halasz F., Schwartz M. *The Dexter Hypertext Reference Model*. Communications of the ACM 37 (2), 30-39, 1994.

[Hardman 93] Hardman L., Bulterman D.C.A., van Rossum G. *The Amsterdam Hypermedia Model: Extending Hypertext to Support Real Multimedia*. Hypermedia 5 (1), 47-69, 1993.

[Hardman 94] Hardman L., Bulterman D.C.A., van Rossum G. *The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model*. Communications of the ACM 37 (2), 50-62, 1994.

[Hardman 99] Hardman, L., van Ossenbruggen, J., Rutledge, L., Bulterman D.C.A. *Hypermedia: The Link with Time*. ACM Computing Surveys, 31 (4), 1999.

[Harmonia UIML] Harmonia, Inc. *User Interface Markup Language (UIML). Draft Specification. Document Version 17 January 2000*. http://www.harmonia.com, 2000.

[Heath 00] Heath, I., Hall, W., Corwder, R., Wills, G., Ballantyne, J. *Towards a new authoring methodology for large-scale hypermedia applications*. Multimedia Tools and Applications 12 (2/3), 2000.

[Hopcroft 79] Hopcroft, J.E., Ullman, J.D. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Series in Computer Science, 1979.

[Horrocks 99] Horrocks I. *Constructing the user interface with statecharts*. Addison-Wesley, 1999.

[Hugh 99] Hugh, C.D. *Hypertext link integrity*. ACM Computing Surveys, 31 (4), 1999.

[IBM Xalan] *IBM Xalan*. http://alphaworks.ibm.com/, 2001.

[IBM Xerces] *IBM Xerces*. http://alphaworks.ibm.com/, 2001.

[IMS] *IMS*. http://www.imsproject.org/, 2001.

[Isakowitz 95] Isakowitz T., Stohr E.A., Balasubramanian P. *RMM: a methodology of structured hypermedia design*. Communications of the ACM 38 (8), 34-43, 1995.

[ISO DSSSL] International Standards Organization. *Document Style Semantics and Specification Language (DSSSL), ISO/IEC IS 10179:1996*. 1996.

[ISO HyTime] International Standards Organization. *Hypermedia/Time-based Structuring Language (HyTime), ISO/IEC IS 10744.1992*. 1992.

[ISO SGML] International Standards Organization. *Standard Generalized Markup Language (SGML), ISO/IEC IS 8879*. 1986.

[Jacobson 00] Jacobson, I., Booch, G., Rumbaugh, J. *El proceso unificado de desarrollo de software*. Addison-Wesley, 2000.

[Kimber a] Kimber W.E. *A Tutorial Introduction to SGML Architectures*. http://www.isogen.com/papers/archintro.html

[Kimber b] Kimber W.E. *Practical Hypermedia: An Introduction to HyTime*. http://www.drmacro.com/bookrev/practhyt/practycalhypermedia.html

[Lee 00] Lee, D., Chu, W.W. *Comparative Analysis of Six XML Schema Languages*. ACM SIGMOD Record, 29 (3),76-87, septiembre 2000.

[Lowe 99a] Lowe, D. B., Bucknell A. J., Webby, R. G. *Improving Hypermedia Development: A Reference Model-Based Process Assessment Method*. The 10th ACM Conference on Hypertext and Hypermedia, Hypertext 99. Darmstadt, 21-28 febrero, 1999.

[Lowe 99b] Lowe D. B., Webby, R. G. *Utilisation of Process Modeling in Improving the Hypermedia Development Process*. The New Review of Hypermedia and Multimedia, 5, 133-150, 1999.

[Maller 96] Maller, E., El Andaloussi, J. *Developing SGML DTDs. From Text to Model to Markup*. Prentice Hall PTR, New Jersey, 1996.

[Mandel 98] Mandel, L., Koch, N., Maier, C. *Extending UML to Model Hypermedia and Distributed Systems*. Technical Report 9804, Institut für Informatik der LMU, 1998.

[Maruyama 00] Maruyama, H., Tamura, K., Uramoto, N. *Sitios Web con XML y Java*. Prentice Hall, 2000.

[Millard 00] Millard, D. E., Moreau, L., Davis, H. C., Reich S. *FOHM: A Fundamental Open Hypertext Model of Investigating Interoperability between Hypertext Domains*. The 11th ACM Conference on Hypertext and Hypermedia, Hypertext 2000, pp 93-102. San Antonio, 30 mayo – 3 junio, 2000.

[MM 98] *María Moliner, Diccionario de uso del español. Segunda edición*. Editorial Gredos. 1998.

[Montero 00] Montero, S., Díaz, P., Aedo, I. *Ariadne: Metodología de Desarrollo para Hipermedia*. IV Jornadas Científicas en Tecnologías de la Información, CINTE 2000, pp 36-43. Cádiz, 23-24 noviembre, 2000.

[Mozilla XUL] Mozilla. *XUL Programmer´s Reference Manual. Fourth Draft*. http://www.mozilla.org/xpfe/xulref/. 2001.

[Muchaluat-Saade 01] Muchaluat-Saade, D.C, Gomes, L.F. *Hypermedia Spatio-Temporal synchronization relations also deserve first-class status*. The 8th International conference on Multimedia Modeling, MMM 2001. Amsterdam, 5-7 noviembre, 2001.

[Murugesan 99] Murugesan, S., Deshpande, Y., Hansen, S., Ginige, A. *Web Engineering: A New Discipline for Development of Web-based Systems*. First ICSE Workshop on Web Engineering, International Conference on Software Engineering. Los Ángeles, 17-18 mayo, 1999.

[Nanard 95] Nanard, J., Nanard, M. *Hypertext design environments and the hypertext design process*. Communications of the ACM, 38 (8), 49-56, 1995.

[Nanard 98] Nanard, J., Nanard, M. *An architecture model for the hypermedia engineering process*. IFIP Working Conference on Engineering of Human Computer Interaction, EHCI'98. Creta, 1998.

[Nanard 99] Nanard, J., Nanard, M. *Toward an Hypermedia Design Patterns Space*. The 10th ACM Conference on Hypertext and Hypermedia, Hypertext 99. Darmstadt, 21-28 febrero, 1999.

[Navarro 98] Navarro A. *Aplicaciones de los lenguajes de marcado en la abstraccion del diseño de un sistema hipermedia*. Trabajo de Tercer Ciclo, Departamento de Sistemas Informaticos y Programacion Universidad Complutense de Madrid, 1998.

[Navarro 00a] Navarro, A., Fernández-Valmayor, A., Fernández, B., Sierra, J.L. *Integration of Markup Languages and Object Oriented Techniques in a Hypermedia Methodology*. The 12th Joint International Conference of the Association for Literary and Linguistic Computing, and the Association for Computer and the Humanities, ALLC-ACH 2000, pp 4-7. Glasgow, 21-25 julio, 2000.

[Navarro 00b] Navarro, A., Fernández, B., Fernández-Valmayor, A., Sierra, J.L. *A Practical Methodology for the Development of Educational Hypermedias*. International Conference on Educational Uses of Information and Communication Technologies, ICEUT 2000, que es parte del 16th IFIP World Computer Congress 2000, Information Processing Beyond Year 2000, pp 217-220. Beijing, 2000.

[Navarro 00c] Navarro, A., Sierra, J.L., Fernández, B, Fernández-Valmayor, A. *XML-based Integration of Hypermedia Design Techniques and Component-Based Techniques in the Production of Educational Applications*. M. Ortega, J. Bravo (Eds) Computers and Education in the 21st Century, pp 229-239. Kluwer Academic Publishers, Holanda, 2000.

[Navarro 00d] Navarro, A., Fernández-Valmayor, A, Fernández B., Sierra, J.L. *La metodología ADDAH en el Dominio Educativo*. Segundo Simposio Internacional de Informática Educativa, SIIE 2000. Puertollano, 15-17 noviembre, 2000.

[Navarro 01a] Navarro, A., Fernández-Valmayor A, Fernández, B., Sierra, J.L. *Using Analysis, Design and Development of Hypermedia Applications in the Educational Domain*. M. Ortega y J. Bravo (Eds), Computers and Education: Towards an Interconnected Society, pp 251-260. Kluwer Academic Publisher, Holanda, 2001.

[Navarro 01b] Navarro, A., Fernández-Valmayor, A., Fernández, B., Sierra, J.L. *Trading Off Multimedia Contents for Adaptation Capabilities: Developing Educational Hypermedia Applications for the Web*. Seventh IFIP World Conference on Computers and Education, WCCE 2001, pp. 157. Copenhague, 29 julio – 3 agosto, 2001.

[Navarro 01c] Navarro, A., Fernández-Valmayor, A., Fernández B., Sierra, J.L. *Desarrollo de aplicaciones hipermedia educativas en la Web*. Tercer Simposio Internacional de Informática Educativa, SIIE 2001. Viseu, 26-28 septiembre, 2001.

[Navarro 02] Navarro, A., Fernández, B., Fernández-Valmayor, A., Sierra, J.L. *Formal-Driven Conceptualization and Prototyping of Hypermedia Applications*. Fundamentals Approaches to Software Engineering 2002, FASE 2002, que forma parte de European joint conferences on Theory and Practice of Software 2002, ETAPS 2002, pp. 308-322. Grenoble, 6-14 abril, 2002. Publicado como Lecture Notes in Computer Science (Vol. 2306), Springer-Verlag, Berlin, 2002.

[Newcomb 91] Newcomb S.R., Neill A. K., Newcomb V.T. *HyTime. The Hypermedia/Time-based Document Structuring Language*. Communications of the ACM 34 (11), 67-83, 1991.

[Nielsen 95] Nielsen, J. *Multimedia and Hypertext. The Internet and beyond*. Academic Press. Morgan Kaufmann Publishers, San Francisco, 1995.

[Niemeyer 00] Niemeyer, P., Knudsen, J. *Curso de Java*. Anaya Multimedia/O´Reilly, 2000.

[Olsina 98] Olsina, L. *Functional View of the Hypermedia Process Model*. The Fifth International Workshop on Engineering Hypertext Functionality at ICSE'98. Kyoto, 20 abril, 1998.

[Oratrix 99] Oratrix. *The GRiNS Player and Editor for SMIL*. Oratrix Development B.V. http://www.oratrix.com/GRiNS/, 1999.

[Paulo 99] Paulo, F. B., Masiero, P. C., Ferreira de Oliveira, M. C. *Hypercharts: Extended Statecharts to Support Hypermedia Specification*. IEEE Transactions on Software Engineering, 25 (1), 33-49, 1999.

[Peat 97] Peat B, Webber, D. *Introducing XML/EDI*. http://www.xmledi-group.org/xmledigroup/start.html, 1997.

[PNML] *Petri Net Markup Language*. http://www.informatik.hu-berlin.de/top/pnml/

[Pressman 93] Pressman, R.S. *A Manager´s Guide to Software Engineering*. McGraw-Hill, Inc.

[Pressman 01] Pressman R.S. *Ingeniería del Software. Un enfoque práctico. Quinta edición*. McGraw-Hill, 2001.

[RealNetworks 99] RealNetworks. *RealSystem G2*. http://www.real.com/g2/, 1999.

[Rossi 99] Rossi, G., Lyardet, F. D., Schwabe, D. *Developing Hypermedia Applications with Methods and Patterns*. ACM Computing Surveys, 31 (4), 1999.

[van Rossum 93] van Rossum, G., Jansen, J., Mullender, K.S., Bulterman, D.C.A. *CMIFEd: A presentation environment for portable hypermedia documents*. First ACM International Conference on Multimedia, pp 183-188. California, 1-6 agosto, 1993.

[Rumbaugh 97] Rumbaugh J., Blaha M., Premerlani W., Hedí F., Lorensen W. *Modelado y diseño orientado a objetos*. McGraw-Hill, 1997.

[Rumbaugh 98] Rumbaugh J., Jacobson I., Booch G. *El lenguaje unificado de modelado. Manual de referencia*. Addison-Wesley, 2000.

[SAX] *SAX*. http://www.saxproject.org/

[Schwabe 95a] Schwabe D., Rossi G., Barbosa S.D.J. *Abstraction, Composition and Lay-Out Definition Mechanisms in OOHDM*. ACM Workshop on Effective Abstractions in Multimedia. San Francisco, 4 noviembre, 1995.

[Schwabe 95b] Schwabe, D., Rossi, G. *The Object-Oriented Hypermedia Design Model*. Communications of the ACM 38 (8), 45-46, 1995

[Schwabe 96] Schwabe D., Rossi G., Barbosa S.D.J. *Systematic Hypermedia Application Design with OOHDM*. The Seventh ACM Conference on Hypertext, Hypertext 96. Washington D.C., 16-20 marzo, 1996.

[Shirota 01] Shirota, Y., Hashimoto, T., Nadamoto, A., Hattori, T., Iizawa, A., Tanaka, K., Sumiya, K. *A TV Program Generation System Using Digest Video Scenes and a Scripting Markup Language*. 34th Hawaii International Conference on System Sciences. Hawaii, 3-6 enero, 2001.

[Sierra 99] Sierra, J.L, Fernández-Valmayor, A, Fernández, B., Navarro, A. *La integración de documentos marcados en aplicaciones hipermedia y su relación con los mecanismos de mappings entre ontologías*. VIII Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA 99. Murcia, 1999.

[Sierra 00a] Sierra, J.L, Fernández, B., Fernández-Valmayor, A., Navarro, A. *Integration of markup languages, document transformations and software components in the development of applications: the DTC approach*. International Conference on Software: Theory and Practice, ICS 2000, pp 191-200. Beijing, 2000.

[Sierra 00b] Sierra, J.L., Fernández, B., Fernández-Valmayor, A., Navarro, A. *Developing XML applications with XML documents, document transformations and software components*. The 10th International Conference on Computers and Information, ICCI 2000. Kuwait, 18-21 noviembre, 2000.

[Sierra 01] Sierra, J.L., Fernández-Valmayor, A., Fernández, B., Navarro, A. *Operationalizing Application Descriptions in DTC: Building Applications with Generalized Markup Tecnologies*. The Thirteenth International Conference on Software Engineering and Knowledge Engineering, pp 379-386. SEKE 2001. Buenos Aires, 13-15 junio, 2001.

[Sommerville 01] Sommerville, I. *Software Engineering*. 6th edition. Addison-Wesley, 2001.

[Sperberg-McQueen 94] Sperberg-McQueen, C.M., Goldstein, R.F. *HTML to the Max A Manifesto for Adding SGML Intelligence to the World-Wide Web*. Second World Wide Web Conference '94: Mosaic and the Web. Chicago, 17-20 octubre, 1994.

[SQLX] *SQLX*. http://www.SQLX.org

[Stotts 89] Stotts P.D., Furuta R. *Petri-Net-Based Hypertext: Document Structure with Browsing Semantics*. ACM Transactions on Office Information Systems 7 (1), 3-29. 1989.

[Stoy 77] Stoy, J.E. *Denotational Sematics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[Studer 99] Studer, R., Fensel, D., Decker, S., Benjamins, V.R. *Knowledge Engineering: Survey and Future directions*. F. Puppe (ed.), Knowledge-based Systems: Survey and future Directions. Lecture notes in Artificial Intelligence (Vol. 1570), Springer-Verlag, 1999.

[Sun Java 2] *Sun Java™ 2 SDK, Standard Edition, Version 1.3.0*. http://java.sun.com, 2000.

[Tompa 89] Tompa F. *A Data Model for Flexible Hypertext Database Systems*. ACM Transactions on Information Systems 7 (1), 85-100, 1989.

[VocML] *Vocabulary Markup Language*. http://orc.dev.oclc.org:5103/nkos/bin00003.bin .

[Walrath 99] Walrath, K., Campione, M. *The JFC Swing Tutorial. A Guide to Constructing GUIs*. Addison-Wesley, 1999.

[Walton 96] Walton, L. *Domain-specific design languages*. http://www.cse.ogi.edu/~walton/dsdls.htm, 1996

[Will 99] Will, U. K., Nürnberg, P. J. *Evolving Hypermedia Middleware Services: Lessons and observations*. 1999 ACM Symposium on Applied Computing (SAC 99), pp 427-436. San Antonio, 28 febrero – 2 marzo, 1999.

[Wills 98] Wills, G. B., Crowder R.M., Heath I., Hall, W. *Industrial Hypermedia Design*. Universidad de Southampton, M98-2 4, 1998.

[WML] *Wireless Markup Language*. http://www.oasis-open.org/cover/wap-wml.html.

[W3C CSS] World Wide Web Consortium. *Cascading Style Sheet, level 2. CSS2 Specification*. http://www.w3.org/TR/REC-CSS2/, 1998.

[W3C DOM] World Wide Web Consortium. *Document Object Model (DOM). Level 1 Specification*. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/, 1998.

[W3C HTML] World Wide Web Consortium. *HTML 4.01 Specification*. http://www.w3.org/TR/html4/, 1999.

[W3C namespaces] World Wide Web Consortium. *Namespaces in XML*. http://www.w3.org/TR/REC-xml-names, 1999.

[W3C schema] World Wide Web Consortium. *XML Schema Part 0: Primer*. http://www.w3.org/TR/xmlschema-0, 2001.

[W3C SMIL] World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL 2.0)*. http://www.w3.org/TR/smil20/, 2001.

[W3C XHMTL] World Wide Web Consortium. *XHTML 1.0: The Extensible HyperText Markukp Language. A Reformulation of HTML 4.0 in XML 1.0*. http://www.w3.org/TR/xhtml1/, 2000.

[W3C XLink] World Wide Web Consortium. *XML Linking Language (XLink) Version 1.0*. http://www.w3.org/TR/2001/REC-xlink-20010627, 2001.

[W3C XMLa] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*. http://www.w3.org/TR/REC-xml, 1998.

[W3C XMLb] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0 (Second Edition)*. http://www.w3.org/TR/2000/REC-xml-20001006, 2000.

[W3C XMLQ] World Wide Web Consortium. *XML Query*. http://www.w3.org/XML/Query

[W3C XPath] World Wide Web Consortium. *XML Path Language (XPath). Version 1.0*. http://www.w3.org/TR/xpath, 1999.

[W3C XPointer] World Wide Web Consortium. *XML Pointer Language (XPointer), W3C Candidate Recommendation 11 September 2001*. http://www.w3.org/TR/xptr, 2001.

[W3C XSL] World Wide Web Consortium. *Extensible Stylesheet Language. Version 1.0*. http://www.w3.org/TR/2001/REC-xsl-20011015/. 2001.

[W3C XSLT] World Wide Web Consortium. *XSL Transformations (XSLT). Version 1.0*. http://www.w3.org/TR/xslt, 1999.

Apéndice A. Pipe n-ario

A.1 Introducción

En este apéndice se proporcionará la reformulación del modelo Pipe descrita en el Capítulo 5 para que soporte convenientemente enlaces n-arios. La idea básica se centra en incluir un número de enlace para cada pareja de contenido y ancla, de tal forma que cuando se seleccione un ancla se activen todos los enlaces que tengan por origen dichos contenidos y ancla, aunque el número de enlace sea distinto. El cambio necesario en la semántica de navegación será el de activar todos los enlaces con origen en dicha ancla.

Otra aproximación sería la de permitir que la relación r se estableciese en conjuntos de contenidos, pero esto presenta el inconveniente de no permitir canalizar enlaces convenientemente. En efecto, si partimos de que los enlaces son:

$$E = C_{ns} \times A \times C \times A$$

Parece razonable caracterizar un enlace binario (por ejemplo) a través de los enlaces con la misma ancla origen: (a, aBC, b, total), (b, aBC, c, total). Pasándolo al formato de la relación r, tendríamos:

$$r: C_{ns} \times A \rightarrow \wp(C \times A)$$
$$(a, aBC) \rightarrow \{ (b, total), (c, total) \}$$

El problema es que en un esquema navegacional como el descrito por la Figura A.1 no se puede canalizar los enlaces tal y como indica la función de asignación de contenidos y de canalización, ya que no podemos elegirlos convenientemente en r(a, aBC).

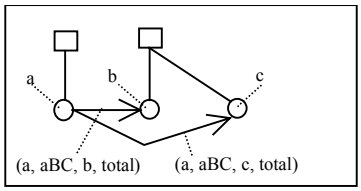

Figura A.1 Esquema navegacional problemático

Por eso es necesario definir:

$$E = C_{ns} \times A \times \mathbb{N} \times C \times A$$

siendo $\mathbb{N}$ el conjunto de números naturales. De esta forma un enlace binario es: (a, aBC, 1, b, total), (a, aBC, 2, c, total). Formalicemos convenientemente esta idea.

A.2 Pipe

La representación de una aplicación hipermedia a través del modelo Pipe [Navarro 02] es una tupla <GC, EN, FC, SN, SP> donde:

- GC: es el *grafo de contenidos*.
- EN: es el *esquema navegacional*.
- FC: son las *funciones de canalización*.
- SN: es la *semántica navegacional*.
- SP: es la *semántica presentacional*.

Para caracterizar una aplicación hipermedia será necesario proporcionar el grafo de contenidos, el esquema navegacional, y las funciones de relación en cada caso, mientras que las semánticas de navegación y presentación son comunes para todas las aplicaciones. Veamos en detalle cada componente del modelo.

A.2.1 Grafo de contenidos

A.2.1.1 Contenidos

Definimos C como el *conjunto de contenidos susceptibles de aparecer en una aplicación hipermedia*. Dividimos este conjunto en dos,

$$C = C_{ns} \cup C_s$$

Donde C_{ns} es el conjunto de *contenidos no* subordinados, y C_s, es *el conjunto de contenidos subordinados*, con $C_{ns} \cap C_s = \varnothing$.

Evidentemente tenemos la necesidad de especificar la existencia de anclas dentro de los contenidos. Con ese fin introducimos el *conjunto de anclas*, A. Para relacionar a los contenidos con sus anclas necesitamos definir la *función de anclaje* a:

$$a: C \rightarrow \wp(A)$$

tal que a un contenido le asigna el conjunto de anclas que tiene asociado. En el caso de contenidos no subordinados estas anclas podrán ser tanto origen como destino de enlaces. En el caso de contenidos subordinados estas anclas solo pueden corresponderse con anclas destino. En particular solo podrá tomar el valor total, que denota la totalidad de un contenido (subordinado o no). Como veremos más adelante, en Pipe no existen *anclas origen* o *anclas destino*, ya que esta no es una característica innata del ancla, sino adquirida en la definición de enlace.

A.2.1.2 Enlaces y relaciones estáticos

Ahora nos encontramos en disposición de definir el *conjunto de enlaces de la aplicación hipermedia estática*, E^0 como:

$$E^0 \subseteq C_{ns}{}^0 \times A^0 \times \mathbb{N} \times C^0 \times A^0$$

Donde:

- $C^0 \subseteq C$ es el *conjunto de contenidos estáticos de una aplicación hipermedia*, con $C^0 = C_{ns}{}^0 \cup C_s{}^0$, $C_{ns}{}^0 \subseteq C_{ns}$, $C_s{}^0 \subseteq C_s$
- $A^0 \subseteq A$ es el *conjunto de anclas de una aplicación hipermedia estática*. Nótese que, $A^0 = \bigcup_{c \in C^0} a(c)$

Veamos que denota un enlace e = (o, ao, n, d, ad) perteneciente al conjunto anterior. La primera coordenada, o, denota el contenido donde se origina el enlace. La segunda coordenada, ao, denota el ancla donde se origina el enlace. La tercera coordenada, n, denota el *número enlace*. La cuarta coordenada, d, denota el contenido destino del enlace. Finalmente, la quinta coordenada, ad, denota el ancla en el contenido destino. Nótese que para representar un enlace n-ario es necesario repetir la información origen n veces. Es decir, si el contenido a enlaza con el b y el c necesitamos en principio los enlaces (a, aBC, 1, b, total), (a, aBC, 2, c, total). Además Pipe solo considera enlaces unidireccionales, ya que la existencia de anclas imposibilita la bidireccionalidad del enlace. En efecto supongamos, retomando el ejemplo de la biografía, que una biografía enlaza con los países europeos de la lista. Deberíamos tener un enlace de la forma (biografía, aEuropa, 1, países, europeos). Si el enlace fuese bidireccional, en enlace de la lista de países con la biografía partiría de la zona de países europeos, cosa bastante poco deseable. Si se desea indicar la existencia de un enlace entre la lista de países y la biografía debería incluirse un enlace de la forma (países, aBiografía, 1, biografía, total), donde el ancla total denota el contenido total, es decir, no especifica un ancla concreta dentro del mismo.

La función de relación

La definición del conjunto E^0 proporcionada en el apartado anterior es demasiado genérica. En última instancia el conjunto E^0 debe denotar una relación binaria entre pares de contenido origen y ancla origen con contenido destino y ancla destino. Para una caracterización más elegante de la semántica navegacional de la aplicación, y para (como veremos en el siguiente apartado) permitir la presencia de contenidos dinámicos, vamos a dividir los enlaces en su parte origen (contenido y ancla) y en su parte destino (contenido y ancla).

Con este fin, y para caracterizar la relación inducida entre contenidos y anclas en base a enlaces, definimos la *función de relación estática*, r,

$$r: C_{ns}^{0} \times A^{0} \times \mathbb{N} \rightarrow C^{0} \times A^{0}$$

tal que si $(c, a, n) \in C_{ns}^{0} \times A^{0} \times \mathbb{N}$, r(c, a, n) es el contenido (y su ancla) con el que se relaciona el contenido c a través del ancla a y el número de enlace n. La función r no es total, ya que en el conjunto anterior hay pares (c, a, n) tq $a \notin a(c)$. No es inyectiva, ya que es posible que dos contenidos ancladas distintos lleven al mismo destino. Finalmente no es suprayectiva, ya que la imagen está contenida en un conjunto demasiado genérico. Así, podemos caracterizar al conjunto E^0, de la siguiente manera:

$$E^{0} = \{ (c, a, n, r(c, a, n)) \mid (c, a, n) \in C_{ns}^{0} \times A^{0} \times \mathbb{N} \}$$

En Pipe el papel de origen o destino de un ancla viene dado por el propio enlace y no por el ancla. Esta aproximación, a otro nivel de abstracción, es similar a la propuesta por HyTime con sus enlaces `hylink` o por XLink con sus enlaces `extended`., tal y como vimos en el Capítulo 4.

Por lo tanto para describir el conjunto de enlaces de una aplicación hipermedia no tenemos más que proporcionar la definición de la función r. Cuando la aplicación sea totalmente estática podremos definir a r por extensión, es decir, proporcionando al conjunto E^0. Cuando tenga componentes dinámicas, la definición de r deberá producirse de manera intensional, tal y como veremos más adelante en el apartado A.2.1.3.

La función de número de enlace

La *función de número de enlace* ne, se encarga de caracterizar los números de enlace que se corresponden para un ancla y contenido concreto. Esta función será necesaria para definir la semántica de navegación de la aplicación.

$$ne: C_{ns}^{0} \times A^{0} \rightarrow \wp(\mathbb{N})$$

$$ne(c, a) = \{ n \mid (c, a, n, r(c, a, n)) \in E^{0} \}$$

Contenido accedido y enlace activado

En base a esta función de relación definimos la función:

$$conAcc: C_{ns}^{0} \times A^{0} \times \mathbb{N} \rightarrow C^{0}$$
$$conAcc(c, a, n) = \prod_{1} r(c, a, n)$$

Donde la función $\prod_i$ representa la proyección de la coordenada i-ésima de una tupla. Esta función caracteriza los contenidos accedidos ($\prod_1 r(c, a, n)$) a través de un enlace ((c, a, n, r(c, a, n))) como consecuencia de la selección de un ancla (a), número de enlace n, en un contenido (c). Nótese que esta función se encarga de caracterizar la composición de las funciones de resolución y acceso del modelo Dexter, siendo en este caso la función de resolución la identidad, y la de acceso la propia función conAcc. Es decir, si:

resolucion: Especificaciones → IDs
acceso: IDs → Componentes

Entonces conAcc = acceso∘resolución

También definimos la función:

$$enlAct: C_{ns}^{0} \times A^{0} \times \mathbb{N} \rightarrow E^{0}$$
$$enlAct(c, a, n) = (c, a, n, r(c, a, n))$$

Esta función caracteriza al enlace activado ((c, a, n, r(c, a, n))) como consecuencia de la selección de un ancla (a), número de enlace n, en un contenido (c). Aunque ahora no tenga mucho sentido la definición de estas dos funciones, nos serán de una utilidad extrema a la hora de definir la semántica navegacional por defecto de la aplicación hipermedia.

A.2.1.3 Enlaces y relaciones dinámicos

Función de generación

La ventaja de caracterizar los enlaces a través de una función de relación r radica en la posibilidad de representar aplicaciones dinámicas. Cuando los contenidos, y enlaces entre estos, sean conocidos a priori, es decir, antes de la ejecución de la aplicación no encontraremos con *aplicaciones hipermedia estáticas* (las vistas hasta ahora). En este caso podremos definir el conjunto de enlaces enumerando por extensión a la función de relación r (es decir, enumerando al conjunto E^0). Por el contrario, cuando sea la aplicación hipermedia la encargada de generar estos contenidos, y los enlaces entre estos, nos encontraremos ante *aplicaciones hipermedia dinámicas*. En este caso para definir el conjunto de enlaces proporcionaremos una descripción intensional de la función de relación r en base a una nueva función, la *función de generación* g. Por lo tanto, en ambos casos (estático o dinámico) la definición del conjunto de enlaces se lleva a cabo exclusivamente sobre la función de relación r. Como la semántica de navegación esta definida sobre la función de relación r, la naturaleza estática o dinámica de la aplicación hipermedia es transparente para dicha semántica navegacional. De aquí la necesidad de definir la función de relación r.

La función de generación se va a encargar de generar un contenido y diversas anclas sobre él en respuesta a la selección de un ancla en un contenido origen. Definimos la *función de generación* g:

$$g: C_{ns} \times A \times \mathbb{N} \rightarrow C \times A \times \wp(A \times \mathbb{N}) \times \wp(A)$$

$$g(c, a, n) = (c', a', AGO, AGD)$$

Donde c´ es el *contenido generado*, a´ es el *ancla destino generada*, AGO es el *conjunto de anclas generadas consideradas como origen* junto con su número de enlace, y AGD es el *conjunto de anclas generadas consideradas como destino* a partir de la selección de un ancla en un contenido ((c, a, n)). La función g no es total, ni inyectiva, ni suprayectiva por las mismas razones por las que no lo era r.

Nótese que la definición de esta función difiere en dos aspectos respecto a la función de generación presentada en el Capítulo 5. El de menor importancia radica en considerar explícitamente las anclas generadas, en vez de recurrir a la función ACO. El segundo, y de mayor enjundia, no caracteriza explícitamente el no determinismo de la función g. Hay dos razones para este cambio. El primero darse cuenta de que si obviamos la posibilidad de que varíe la información generada por dicha función con el paso del tiempo (es decir, si abstraemos el hecho de que una base de datos puede cambiar con el tiempo) eliminamos gran parte del indeterminismo. La otra fuente de indeterminismo viene de considerar que el mismo contenido puede generar diversos resultados. Esto se debe básicamente a que abstraemos "en exceso", y no consideramos que un contenido formulario genera la información de respuesta no en base al formulario, si no a la *entrada* que se da en el formulario. Considerando un parámetro más:

$$g: Entrada \times C_{ns} \times A \times \mathbb{N} \rightarrow C \times A \times \wp(A \times \mathbb{N}) \times \wp(A)$$

la función sería determinista. También habría que redefinir r, pero en este caso la entrada podría ser un valor $\perp$. En cualquier caso, hemos optado por no considerar este parámetro porque no intentamos dar una caracterización *absoluta* de la aplicación hipermedia. Es decir, no intentamos considerar todos los enlaces y/o contenidos susceptibles de aparecer en determinada aplicación. Solamente proporcionamos una notación de diseño (conceptualización, mejor dicho). La interpretación de dicha notación solo cobra sentido desde el punto de vista hipermedia, es decir, a partir de la selección de un enlace. Precisamente esto es lo que indica la semántica de navegación proporcionada, aunque a nivel local y no global (es decir, cada vez que se selecciona un enlace *concreto* y no todos los enlaces *posibles*).

Ahora podemos definir la función de relación y la función de anclaje en base a la función g. En efecto,

$$r(c, a, n) = (c', a') = \prod_{12} g(c, a, n)$$
$$a(c') = \prod_1 AGO \cup AGD = \prod_1 \prod_3 g(c, a, n) \cup \prod_4 g(c, a, n)$$

Es decir, en aplicaciones dinámicas los contenidos y anclas ((c, a, n)) se relacionan con los contenidos y anclas generadas dinámicamente (r(c, a, n)) a través de la función de generación ($\prod_{12} g(c, a, n)$). Además ahora las anclas de un contenido generado dinámicamente (a(c´)) viene proporcionada por la función de generación ($\prod_1 \prod_3 g(c, a, n) \cup \prod_4 g(c, a, n)$). Evidentemente debe imponerse la restricción ($c' \in AGD \wedge total \in AGD$). Nótese

que en contenidos generados, las anclas siguen sin tener naturaleza de origen o destino. La distinción entre los conjuntos AGO y AGD se hace necesaria porque ahora se debe determinar si las anclas son origen o destino de un enlace de manera automática, y precisamente su división en dos conjuntos (no necesariamente disjuntos) permite esta caracterización.

Por tanto, la función de generación g sirve para caracterizar los contenidos y enlaces dinámicamente generados. Como ya comentamos en su definición, la función de relación r sirve para unificar la definición de los enlaces, haciendo de caja negra respecto a enlaces y contenidos estáticos o dinámicos. Esto va a ser importante en la definición de la semántica de navegación de Pipe. De hecho, las funciones de contenido accedido (conAcc) y enlace activado (enlAct) están definidas en términos de la función r, obviándose en la semántica de navegación la naturaleza estático y/o dinámica de los contenidos y enlaces.

Es decir, en el caso de contenidos y enlaces generados dinámicamente tenemos que:

$$conAcc(c, a, n) = \prod_1 r(c, a, n) = \prod_1 g(c, a, n)$$
$$enlAct(c, a, n) = (c, a, n, r(c, a, n)) = (c, a, n, \prod_{12} g(c, a, n))$$

A.2.1.4 Aplicación de Pipe

Vamos a ver ahora una serie de conceptos que son necesarios para aplicar Pipe en casos prácticos.

Contenido generado, enlace generado y contenido anclado

Con el fin de aligerar la notación y hacerla más clara definiremos una serie de funciones que no son más que proyecciones de la función g.

Definimos la *función de contenido generado*, cg,

$$cg: C_{ns} \times A \times \mathbb{N} \rightarrow C$$

$$cg(c, a, n) = c' = \prod_1 g(c, a, n)$$

Esta función se encarga de caracterizar al contenido generado como consecuencia de la activación de un ancla (a), con número n, en un contenido (c).

La segunda función es la *función de enlace generado*, eg,

$$eg: C_{ns} \times A \times \mathbb{N} \rightarrow E$$

$$eg(c, a, n) = (c, a, n, c', a') = (c, a, n, \prod_{12} g(c, a, n)) = (c, a, n, r(c, a, n))$$

donde E denota al conjunto de todos los enlaces posibles, es decir,

$$E = C_{ns} \times A \times \mathbb{N} \times C \times A$$

Esta función se encarga de caracterizar al enlace generado ((c, a, n, r(c, a, n))) como consecuencia de la activación de un ancla (a), con número de enlace n, en un contenido(c).

Como ya hemos comentado, estas funciones no introducen ningún concepto nuevo, y simplemente nos ayudarán a aplicar en la práctica al modelo. De hecho tenemos que si no consideramos un contenido, ancla y número que se corresponda con el origen de un enlace estático inicial:

$$cg(c, a, n) = \prod_1 g(c, a, n) = \prod_1 r(c, a, n) = conAcc(c, a, n)$$
$$eg(c, a, n) = (c, a, \prod_{12} g(c, a, n)) = (c, a, r(c, a, n)) = enlAct(c, a, n)$$

A pesar de que en la práctica tenemos esta coincidencia de funciones hemos decidido utilizar dos nombres distintos para el mismo concepto por el contexto de uso. La terminología contenido accedido y enlace activado tiene sentido en la semántica navegacional, lugar donde poco importa si los contenidos y enlaces son

estáticos o dinámicos. Precisamente con ese fin definimos a la función de relación r. La terminología contenido generado y enlace generado tiene sentido en el proceso de asignar contenidos y enlaces entre estos a partes del esquema navegacional (veremos esto más adelante), y en este contexto si resulta interesante distinguir entre contenidos y enlaces estáticos y dinámicos.

Es decir, ahora el contenido accedido por la selección de un enlace de contenidos (conAcc(c, a, n)) es el contenido generado por la función g (cg(c, a, n)), y el enlace activado en dicho proceso (enlAct(c, a, n)) es el generado por la función g (eg(c, a, n)). Al tener una definición para las funciones de contenido accedido y enlace activado (conAcc y enlAct) tanto en el caso de aplicaciones estáticas como en el caso de aplicaciones dinámicas podemos utilizarlas para unificar la semántica de navegación que veremos en este capítulo.

Aun necesitamos una tercera función que nos será útil para aplicar el modelo, la *función de contenido anclado*, ca,

$$ca: C \rightarrow \wp(C_{ns} \times A \times \mathbb{N})$$

$$ca(cg(c, a, n)) = \{ (c', p, n) \mid c' = \prod_1 g(c, a, n), a' \in \prod_3 g(c, a, n) \}$$

Esta función se encarga de caracterizar los contenidos anclados que se generan como resultado de la aplicación de la función de generación g.

Contenidos generados, enlaces generados, contenidos anclados

En la práctica estas funciones se aplican tanto a elementos como a conjuntos de elementos, por esta razón vamos a extender su definición. En efecto, sea $CA \subseteq C_{ns} \times A \times \mathbb{N}$ definimos,

La *función de contenidos generados*, cg,

$$cg: \wp(C_{ns} \times A \times \mathbb{N}) \rightarrow \wp(C)$$

$$cg(CA) = \{ \prod_1 g(c, a, n) \mid (c, a, n) \in CA \}$$

La segunda función es la *función de enlaces generados*, eg,

$$eg: \wp(C_{ns} \times A \times \mathbb{N}) \rightarrow \wp(E)$$

$$\begin{aligned} eg(CA) &= \{ (c, a, n, c', a') \mid (c', a') \in \prod_{12} g(c, a, n), (c, a, n) \in CA \} \\ &= \{ (c, a, n, r(c, a, n)) \mid (c, a, n) \in CA \} \end{aligned}$$

La tercera función es la *función de contenidos anclados*, ca,

$$ca: \wp(C) \rightarrow \wp(C_{ns} \times A \times \mathbb{N})$$

$$ca(cg(CA)) = \{ (c', p, n) \mid c' = \prod_1 g(c, a), (p, n) \in \prod_3 g(c, a, n), (c, a, n) \in CA \}$$

Contenidos y enlaces de aplicaciones dinámicas

Ahora estamos en condiciones de definir cuales son los contenidos de la aplicación hipermedia y sus enlaces en base a la función de generación g. De esta forma si hay $N \geq 1$ aplicaciones de la función de generación tenemos que, los contenidos de la aplicación C_A, y los enlaces de la misma E_A son:

$$C_A = C^N$$
$$E_A = E^N$$

siendo:

$$C^k = C^{k-1} \cup cg(CA^{k-1})$$
$$E^k = E^{k-1} \cup eg(CA^{k-1})$$
$$CA^k = ca(cg(CA^{k-1}))$$

para $1 \leq k \leq N$, C^0 *los contenidos estáticos*, E^0 los *enlaces estáticos*, y CA^0 los *contenidos anclados iniciales*. Estos contenidos anclados iniciales son los contenidos que iniciarán la primera aplicación de la función de generación g. Es decir, los contenidos y enlaces de la aplicación son los estáticos más los generados en sucesivas iteraciones de la función de generación g.

Además, debemos exigir una consistencia en las anclas de los contenidos ya existentes, respecto a los que se generan. Por tanto debemos indicar que:

$$\forall(c, a, n)(\Pi_1 g(c, a, n) \in C^0 \Rightarrow \Pi_3 g(c, a, n) = \varnothing \ \wedge \Pi_4 g(c, a, n) = \{ \text{total} \})$$

Esta restricción indica que si generamos un contenido estático no podemos modificar las anclas que sirven para generar contenidos dinámicos (ni ninguna otra).

También debemos indicar que no podemos generar el mismo contenido dinámico dos veces, es decir:

$$\forall k \ 1 \leq k \leq N \ ((cg(CA^{k-1}) \cap (C^{k-1} \backslash C^0)) = \varnothing)$$

También debemos exigir que si generamos un contenido subordinado este no puede contener anclas consideradas como origen.

$$\forall(c, a, n)(\Pi_1 g(c, a, n) \in C_s \Rightarrow \Pi_3 g(c, a, n) = \varnothing \ \wedge \Pi_4 g(c, a, n) = \{\text{total}\})$$

Como la función de generación se aplica N veces tenemos que:

$$\forall(c, a, n) \in CA^{N-1} \ (\Pi_3 g(c, a, n) = \varnothing \ )$$

Nótese además que la función g debe *reinicializarse* si volvemos a acceder a un elemento que originó la generación de contenidos (es decir un elemento en CA^0). De lo contrario solo podríamos caracterizar una generación, o la propia función g no sería función, ya que al mismo origen le podrían corresponder dos imágenes distintas (no olvidemos que *g* es no determinista). Dicha reinizialización consistiría en borrar todos los nodos y enlaces generados a partir de los contenidos anclados iniciales CA^0. De esta forma volveríamos a considerar los conjuntos C^0, E^0 y CA^0.

Respecto a las anclas tenemos que en el caso de los contenidos dinámicos, es decir, si $c' \in C_A \backslash C_0$

$$a(c') = AGO \cup AGD = \prod_3 g(c, a, n) \cup \prod_4 g(c, a, n)$$

Nótese que en contenidos generados, las anclas siguen sin tener naturaleza de origen o destino. La distinción entre los conjuntos AGO y AGD se hace necesaria porque ahora se debe determinar si las anclas son origen o destino de un enlace de manera automática, y precisamente su división en dos conjuntos (no necesariamente disjuntos) permite esta caracterización.

Por último, *el conjunto de anclas de la aplicación* $A_A$sería,

$$A_A = \bigcup_{c \in C_A} a(c)$$

A.2.2 Esquema navegacional

N: los nodos del esquema navegacional

Definimos el *conjunto de nodos del esquema navegacional*, N como:

$$N = N_c \cup N_x \cup N_a$$

donde,

- N_c: es el conjunto de *nodos contenedores*, con $N_c \cap N_x = \varnothing$, $N_c \cap N_a = \varnothing$

- N_x: es el conjunto de *nodos nexos*, con
  $N_x \cap N_c = \varnothing$, $N_x \cap N_a = \varnothing$
- N_a: es el conjunto de *nodos activadores de nexos*, con
  $N_a \cap N_x = \varnothing$, $N_a \cap N_c = \varnothing$

Los nodos pueden ser de tres tipos distintos: contenedores, nexos y activadores de nexos. Los *nodos contenedores* son aquellos que sirven de soporte para los contenidos de la aplicación hipermedia, es decir, son los nodos que tienen asignados contenidos. Es importante darse cuenta que los nodos contenedores pueden contener información de muy diverso tipo como, por ejemplo, texto con hiperenlaces, imágenes, un clip de audio o un proceso subordinado. Desde un punto de vista navegacional son equivalentes a paneles que aparecen dentro de ventanas.

Los nodos nexos, son los encargados de representar a las ventanas de la aplicación. Estos nodos funcionan como enumeradores de los nodos de contenidos. Es decir, los nodos nexo son los encargados de proporcionar contexto al modelo Pipe.

Los nodos activadores de nexos sirven para activar nodos nexo desde el esquema navegacional directamente, con total independencia de los contenidos (veremos en la semántica navegacional como es posible activar un nodo nexo desde un contenido asignado a un nodo contenedor). Los nodos nexo se representan con un cuadrado, los nodos contenedor con un círculo, y los nodos activadores de nexos con un triángulo. La Figura A.2 ilustra esta notación.

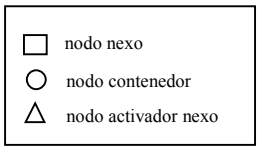

Figura A.2 Representación visual de los nodos del

A partir de ahora en adelante cuando a un nodo lo denotemos por c_i estaremos indicando que $c_i \in N_c$. Cuando a un nodo lo denotemos por x_i estaremos indicando que $x_i \in N_x$. Cuando a un nodo lo denotemos por a_i estaremos indicando que $a_i \in N_a$. Finalmente, cuando ponemos n_i, estamos diciendo que $n_i \in (N_x \cup N_c)$.

A: el conjunto de arcos del esquema navegacional

Los arcos establecidos entre los nodos del esquema navegacional representan las relaciones existentes entre las ventanas, paneles y activadores de ventanas de la aplicación. Así definimos el *conjunto de arcos del esquema navegacional*, A:

$$A \subseteq (N \times N \times P) \cup (N \times N \times P_s \times \mathbb{R})$$

donde,

N: es el conjunto de nodos de la aplicación
P: es el tipo de arco (no sincro) entre los nodos, siendo P = { conexión, enlace }
P_s: es el tipo de arco sincro entre los nodos, siendo P_s = { conexiónS, enlaceS }
$\mathbb{R}$: el conjunto de números reales, indicando la información temporal para los enlaces sincro

En cuanto a la representación visual de los distintos arcos tenemos:

conexión:línea continua
conexiónS: línea continua decorada con la información temporal
enlace: flecha
enlaceS: flecha decorada con la información temporal

La Figura A.3 recoge dicha representación gráfica.

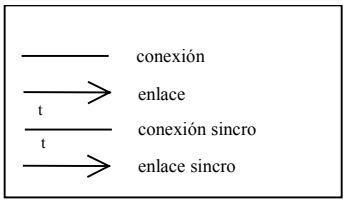

Figura A.3 Representación visual de los arcos del modelo

Además los elementos de A deben cumplir,

Si $a \in A$, entonces a debe ser de la forma

- $a = (x_i, x_j, \text{enlaceS}, t) \wedge \forall k((x_i, x_k, \text{enlaceS}, t) \in A \Rightarrow k=j)$
- $a = (x_i, c_j, \text{conexión}) \wedge (x_i, c_j, \text{conexiónS}, t) \notin A$
  $\wedge \neg\exists k\ ((k \neq i) \wedge ((x_k, c_j, \text{conexión}) \in A \vee (x_k, c_j, \text{conexiónS}, t) \in A))$
- $a = (x_i, c_j, \text{conexiónS}, t) \wedge (x_i, c_j, \text{conexión}) \notin A$
  $\wedge \neg\exists k\ ((k \neq i) \wedge ((x_k, c_j, \text{conexión}) \in A \vee (x_k, c_j, \text{conexiónS}, t) \in A))$
- $a = (x_i, b_j, \text{conexión}) \wedge \forall k((x_k, b_j, \text{conexión}) \in A \Rightarrow k=i)$
- $a = (c_i, c_j, \text{enlace}) \wedge \exists r(x_r, c_i, \text{conexión}) \in A \wedge \exists s(x_s, c_j, \text{conexión}) \in A$
- $a = (b_i, x_j, \text{enlace}) \wedge \forall k((b_i, x_k, \text{enlace}) \in A \Rightarrow k=j)$

Es decir las conexiones permitidas en A quedan recogidas en la Tabla A.1 y en la Figura A.4.

origen\destino	**nodo nexo**	**nodo contenedor**	**n. activador n.**
nodo nexo	enlace sincro *(único)*	conexión, conexión sincro *(conectado a un único nexo)*	conexión *(conectado a un único nexo)*
nodo contenedor	-	enlace	-
n. activador n.	enlace *(único)*	-	-

Tabla A.1 Relaciones posibles entre los nodos del esquema navegacional

Es decir, si los nodos nexo representan ventanas, los contenedor paneles (virtuales o no) de esas ventanas, y los activadores de nexos botones tenemos lo siguiente. Un *enlace sincro entre dos nodos nexo* denota una relación de sincronización entre dos nodos nexo (ventana). Una *conexión entre un nodo nexo y un nodo contenedor* denota una relación de agregación, por la cual el nodo nexo (ventana) contiene al nodo contenedor (panel). Una *conexión sincro entre un nodo nexo y un nodo contenedor* es una conexión con información sobre sincronización. Una *conexión entre un nodo nexo y un nodo activador de nexo* denota una relación de agregación entre el nodo nexo (ventana) y el nodo activador de nexo (botón). Un *enlace entre un nodo contenedor y otro nodo contenedor* denota una relación navegacional entre nodos contenedor (paneles). Un *enlace entre un nodo activador de nexo y un nodo nexo* denota una relación de activación no temporal entre nodos nexo (ventanas). Nótese que tanto en la Tabla A.5 como en las expresiones que la preceden, la relación de conexión entre un nodo nexo y nodos contenedor o activador de nexo es unidireccional. Aunque dicha relación no tiene ninguna dirección, hemos preferido mantenerla unidireccional para poder utilizar una notación más compacta. En contraposición, la relación de enlace si es direccional.

También debemos indicar que todo nodo nexo debe tener conectado a un nodo contenedor, o a un activador de nexo, que todo nodo contenedor debe estar conectado a una ventana, y que todo activador de nexo debe estar conectado a una ventana y activar a otra. Las siguientes restricciones lo garantizan.

- $\forall\ x_i \exists a \in A\ (x_i = \Pi_1 a \wedge \Pi_3 a \in \{\text{conexión, conexiónS}\})$
- $\forall\ c_i \exists a \in A\ (c_i = \Pi_2 a \wedge \Pi_3 a \in \{\text{conexión, conexiónS}\})$
- $\forall\ a_i \exists a \in A\ \exists b \in A((a_i = \Pi_2 a \wedge \Pi_3 a \in \{\text{conexión, conexiónS}\}) \wedge (a_i = \Pi_1 b))$

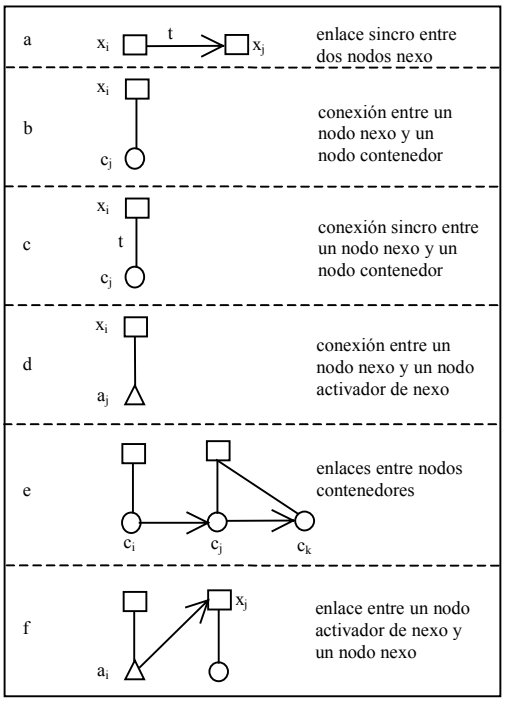

Figura A.4 Relaciones posibles entre los nodos del modelo

En la figura anterior podemos considerar tres categorías de relaciones entre los elementos del esquema navegacional:

- *Relaciones estructurales*. Son aquellas establecidas por conexiones y conexiones sincro entre un nodo nexo y nodos contenedores y activadores de nexos. Casos b, c y d.
- *Relaciones navegacionales independientes de los contenidos*. Son aquellas que establecen un cambio de nodo nexo en función de la existencia de un enlace sincro o de la activación de un activador de nexo. Casos a y f.
- *Relaciones navegacionales que canalizan las inducidas por los contenidos*. Son aquellas que se establecen por enlaces entre nodos contenedores del mismo o de distinto nodos nexos. Caso e. Adelantando acontecimientos estas relaciones las denominaremos *tuberías*.

En efecto, al conjunto de enlaces entre nodos contenedores lo denominaremos conjunto de *Tuberías*, T,

$$T = \{ (c_i, c_j, \text{enlace}) \in A \}$$

Dichas conexiones se denominan de esta forma porque ellas serán las encargadas de *canalizar* los enlaces entre contenidos a nivel navegacional. Es decir, si tenemos un enlace entre una biografía y una foto de España (contenidos) y queremos indicar que cuando se seleccione dicho enlace tanto la biografía como la foto deben permanecer en dos paneles distintos de una misma ventana, no tendremos más que definir una tubería entre dos nodos contenedor de un mismo nodo nexo, y canalizar el enlace de contenidos por dicha tubería. De esta forma la Figura A.5 representa una ventana de la aplicación en la que aparece un nodo nexo (ventana). De este nodo cuelgan dos nodos contenedores (paneles). Además vemos como existe una tubería los nodos contenedor.

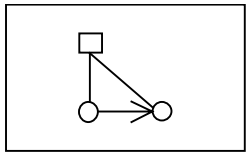

Figura A.5 Nodos y arcos en el modelo

Con respecto a las conexiones y enlaces sincronizados, la etiqueta *t* de tiempo puede ser un valor entero, o una función que calcula el tiempo en función de las relaciones temporales del nodo sincro con respecto al resto de nodos contenedores del mismo nodo nexo. Es decir, supongamos:

$$a \in A,\ a = (\ x_i, c_j, \text{conexiónS}, h(c_k)) \wedge ((x_i, c_k, \text{conexión}) \in A \vee (x_i, c_k, \text{conexiónS}, t) \in A\ )$$

Entonces en función de h() tenemos:

- h() = alComienzo: $N_c \rightarrow \mathbb{R}$, alComienzo($c_k$) indica que el nodo c_j empezará a reproducirse cuando empiece a reproducirse el nodo c_k. Nótese que de esta forma las conexiones entre un nodo nexo y sus nodos contenedores pueden interpretarse como una extensión de la función anterior, donde los nodos contenedores empiezan a reproducirse al comienzo de su nodo nexo.
- h() = alFinal(): $N_c \rightarrow \mathbb{R}$, alFinal($c_k$) indica que el nodo c_j empezará a reproducirse cuando termine de reproducirse el nodo c_k.
- h() = respectoAlComienzo: $N_c \times \mathbb{R} \rightarrow \mathbb{R}$, respectoAlComienzo($c_k$, t) indica que el nodo c_j empezará a reproducirse t segundos después del comienzo del nodo c_k.
- h() = respectoAlFinal: $N_c \times \mathbb{R} \rightarrow \mathbb{R}$, respectoAlFinal($c_k$, t) indica que el nodo c_j empezará a reproducirse t segundos antes del final del nodo c_k.

Nótese que todas las funciones producen como resultado un real, indicando el tiempo con respecto al comienzo del nodo nexo cuando se debería empezar a reproducir los contenidos. Por otro lado si en vez de tener una conexión sincronizada tenemos un enlace sincro entre nodos contenedores, las funciones son igual, es decir, supongamos:

$$a \in A,\ a=(x_i, x_j, \text{enlaceS}, h(c_k)) \wedge ((x_i, c_k, \text{conexión}) \in A \vee (x_i, c_k, \text{conexiónS}, t) \in A$$

entonces h(), puede ser cualquiera de las funciones del caso anterior.

Con estas funciones y la semántica navegacional que describiremos a continuación, el modelo Pipe es capaz de representar relaciones de sincronización avanzadas como las descritas en [Hardman 99].

A.2.3 Funciones de canalización

d: la función de asignación de contenidos

La función de asignación de contenidos es la encargada de asignar contenidos a los nodos contenedor. Definimos la *función de asignación de contenidos*, d:

$$d: N_c \rightarrow (C^0 \cup \{\text{null}\}) \times \wp(C)$$

De esta forma a cada nodo contenedor se le asigna un contenido por defecto (o el valor null, si no hay contenido por defecto), y un conjunto de contenidos. El valor por defecto es el contenido que el nodo contenedor debe mostrar cuando sea activado. El conjunto de contenidos son los posibles contenidos que el nodo podrá mostrar a lo largo de la ejecución de la aplicación. Nótese que por definición la función d es total, ya que no tiene sentido considerar espacio para contenidos inexistentes. Además no tiene porque ser inyectiva, ni suprayectiva. No es inyectiva porque es posible que asignemos los mismos contenidos a dos nodos distintos. Respecto a no ser suprayectiva, hay que darse cuenta que razonamos en términos de $\wp(C)$ y no parece razonable esperar que existan nodos contenedores para todas las posibles combinaciones de contenidos. Si razonásemos en términos de C, tendría que serlo, ya que no tendría sentido el disponer de contenidos que no se asignan a ninguna presentación. Esta función presenta una estrecha relación con la función de canalización de enlaces que veremos a continuación. Además la función no se define en sentido inverso debido ya que al no ser inyectiva no obtendríamos una función.

Además imponemos la condición de que el contenido por defecto de un nodo contenedor sincro no puede contener enlaces, es decir,

$$\forall c_j(x_i, c_j, \text{conexiónS}, t) \in A\ (a(\prod_1 d(c_j)) = \{\text{total}\})$$

l: la función de canalización de enlaces

La función de canalización de enlaces se encarga de relacionar los enlaces existentes a nivel de navegación o tuberías, T, con los enlaces existentes a nivel de contenidos, representados por $C_{ns} \times A \times C \times A$. De esta forma definimos la *función de canalización de enlaces*, l:

$$l: T \rightarrow \wp(E)$$

Además, l debe cumplir $\forall$ (c, a, n, r(c, a, n))$\in$ E,

- $((c, a, n, r(c, a, n)) \in l(c_i, c_j, enlace) \wedge (c, a, n, r(c, a, n)) \in l(c_i, c_k, enlace)) \Rightarrow j = k$
- $(c, a, n, r(c, a, n)) \in l(c_i, c_j, enlace) \Rightarrow c \in d_{1\cup2}(c_i) \wedge conAcc(c, a, n) \in d_{1\cup2}(c_j)$
- $\forall m \forall n((c, a, m, r(c, a, m)) \in l(c_i, c_j, enlace) \wedge (c, a, n, r(c, a, n)) \in l(c_i, c_j, enlace) \Rightarrow m=n)$
- $\exists k \forall m (m \in ne(c, a) \Rightarrow$
  $\exists j ((c_i, c_j, enlace) \in A \wedge enlAct(c, a, m) \in l(c_i, c_j, enlace) \wedge (x_k, c_j, conexión) \in A))$

donde $d_{1\cup2}(c) = \{d_1(c)\} \cup d_2(c) = \{\prod_1 d(c)\} \cup \prod_2 d(c)$

La primera condición garantiza que el mismo enlace de contenidos no puede estar asignado a dos tuberías distintas que parten del mismo nodo. La razón es bien sencilla, ya que cuando se active el enlace en el nodo debemos ser capaces de determinar por que tubería se va a canalizar (veremos esto más adelante). La segunda condición garantiza que los contenidos origen y los contenidos destino del enlace de contenidos estén asignados como contenidos de los nodos contenedor origen y destino de la tubería que canaliza dicho enlace. La tercera condición garantiza que dos mismos destinos de un enlace n-ario estén canalizados por la misma tubería. De no ser así se podrían producir colisiones todos los destinos de los enlaces n-arios De esta forma todos los contenidos destino del enlace n-ario tendrán un nodo contenedor asignado, tal y como asegura la segunda condición.

La función de canalización determina por que enlace navegacional van a pasar los diversos enlaces a nivel de contenidos, por eso razonamos con $\wp(E)$, y no con E, ya que es posible que la misma tubería canalice más de un enlace. Esta función es una de las partes más complejas del modelo Pipe y por eso le prestaremos una especial atención. El modelo parte de una escrupulosa separación entre el nivel de contenidos y sus enlaces, y entre el nivel de navegación y sus enlaces. Con esto se obtiene el beneficio de poder razonar a dos niveles ortogonales: contenidos y navegación de los mismos. Esta ventaja presenta el inconveniente de tener que ligar contenidos a su presentación, y en concreto a como vamos a interpretar los enlaces que poseen los contenidos. Por ejemplo, pensemos en el sitio Web del profesor. En este caso nos restringiremos a la biografía, la información sobre los estudios y la docencia. Aunque existen enlaces entre la biografía y los estudio y docencia, la existencia de dichos enlaces no determina como van a ser navegados. Una posibilidad es que la biografía aparezca en el lado izquierdo de la ventana, y cuando seleccionamos la facultad o carrera, la descripción sobre esta aparece en el lado derecho de la misma ventana. Otra opción es que en una ventana aparezca la biografía y cuando seleccionamos y cuando seleccionamos la facultad o carrera, la descripción sobre esta aparece en otra ventana.

La función l es total, pero no es inyectiva ni suprayectiva. Es total porque no tiene sentido considerar tuberías que no van a canalizar enlaces. No es inyectiva ya que aunque cada tubería solo puede canalizar un conjunto de enlaces de contenido, estos pueden estar asignados a tuberías diferentes (lo que representa distintas interfaces para los mismos contenidos). Veremos este supuesto más adelante en un ejemplo. No es suprayectiva al trabajar con $\wp(E)$. De trabajar solo con E sería suprayectiva ya que deberíamos asignar todos los enlaces a nivel de contenidos a alguna tubería. Además, esta función es total, ya que no tiene sentido asignar contenedores de información que no van a contener ninguna información. Además la función no se define en sentido inverso ya que al no ser inyectiva no sería función.

p: la función de presentación para los nodos y contenidos

La estructura de nodos nexo y nodos contenedores son una abstracción de ventanas formadas por paneles, dentro de las cuales aparecen contenidos y enlaces. Este es el denominado esquema navegacional, ya que indica que componentes aparecen, en que contexto, y con que enlaces. Dicho esquema no determina la

posición concreta de los paneles dentro de las ventanas, ni la fuente, color o estilo del texto, etc. Pues bien las especificaciones de presentación se encargan de determinar estos factores. Es decir, la función de presentación es una abstracción de los parámetros de presentación de una interfaz gráfica actual (al igual que la semántica de navegación es una abstracción de la interfaz de usuario). Para asignar especificaciones de presentación a cada nodo utilizamos la función p, en efecto:

$$p: N \rightarrow EP$$

donde,

N: es el conjunto de nodos
EP: es el conjunto de especificaciones de presentación para cada nodo.

Nótese que los nodos funcionan como contenedores abstractos de información, indicando únicamente que contenidos aparecen dentro de que ventanas, y que enlaces parten de estos contenidos. Es decir solo determinan la estructura navegacional de la aplicación. Para fijar la estructura presentacional completa necesitamos la función p y las especificaciones de presentación.

Tal y como hemos definido la función p, esta se encarga de determinar la posición espacial de los nodos del modelo, y una presentación por defecto para los contenidos. También podemos considerar una presentación especial para los contenidos que tuviese en cuenta la organización estructural de estos. En este caso también llamamos a esta función p. En efecto,

$$p: C \rightarrow EPC$$

En este caso a cada contenido se le aplica una especificación de presentación de contenido, cuya complejidad puede variar según la naturaleza de los mismos. Por ejemplo, en casos concretos se podrían proporcionar especificaciones de presentación similares a hojas de estilo CSS o similares. También es importante darse cuenta que los nodos contenedor pueden tener asignados procesos subordinados. En este caso podrían proporcionarse construcciones similares a las propuestas en [Horrocks 99].

Hemos obviado una caracterización en mayor detalle de esta función ya que su utilidad es parcial en labores de conceptualización. De todas formas se incluye para mostrar su viabilidad de uso en diseño. Nótese por tanto, que con las funciones de presentación no se intenta capturar todas las posibilidades de una interfaz gráfica de usuario actual. Simplemente se pretende dar un soporte básico a las mismas. La razón se debe al uso de Pipe como herramienta de conceptualización y prototipado. En la medida que se desease utilizar Pipe como herramienta de diseño y construcción (induciendo por tanto una metodología) sería deseable ampliar la potencia expresiva de estas funciones de presentación.

A.2.4 Semántica de navegación y semántica de presentación

SN: la semántica navegacional del modelo

Utilizando la formalización estática del modelo vamos a definir la semántica de presentación del modelo, es decir, que cambios se suceden en la interfaz cuando se activa un enlace de contenidos. Para esto introduciremos los conceptos de nodo activo, activación de un nodo nexo, y activación de un hiperenlace.

$$SN = (a, f)$$

donde,

- a: es la función de activación de un nodo nexo
- f: es la función de activación de un hiperenlace

Antes de nada definiremos el concepto de *nodo activo*. Un nodo activo no es más que un nodo que está mostrando sus contenidos por ventana (si es un nodo contenedor), o un nodo que se hace visible al usuario (activador de nexo o ventana). Los caracterizaremos como un simple conjunto, y los representaremos por un punto dentro del nodo, o marcando el nombre del nodo activo en negrita (en este texto optaremos por la segunda alternativa).
Para determinar que sucede dentro del modelo hipermedia cuando se navega por él vamos a determinar primero que partes del modelo debemos tener en cuenta y porque.

- Activos $\subseteq V = N \cup (N \times \mathbb{R})$. Indica en cada momento cual es el conjunto de nodos activos, así como información temporal.
- Mostrar $\subseteq M = (N_c \times C) \cup (N_c \times C \times \mathbb{R})$. Indica que contenidos exhibe cada nodo contenedor del nodo nexo activo, así como información sobre sincronización si fuera necesario.

Lo primero de todo debemos indicar que el inicio de la navegación se va a producir por un nodo nexo determinado (es decir, por una ventana determinada). Este hecho se corresponde con activar el nodo nexo inicial. Para activar nodos nexo disponemos de la función de activación a,

$$a: N_x \rightarrow \wp(V) \times \wp(M)$$

de tal forma que $a(x_i) = (Activos_n, Mostrar_n)$

donde,

$$\begin{aligned} Activos_n = \{x_i\} &\cup \{ c_j \mid (x_i, c_j, \text{conexión}) \in A \} \\ &\cup \{ a_j \mid (x_i, a_j, \text{conexión}) \in A \} \\ &\cup \{ c_j \mid (x_i, c_j, \text{conexiónS}, t) \in A \} \\ &\cup \{ (x_j, t) \mid (x_i, x_j, \text{enlaceS}, t) \in A \} \end{aligned}$$

$$\begin{aligned} Mostrar_n = &\{ (c_j, d_1(c_j)) \mid (x_i, c_j, \text{conexión}) \in A \} \\ &\cup \{ (c_j, d_1(c_j), t) \mid (x_i, c_j, \text{conexiónS}, t) \in A \} \end{aligned}$$

Es decir, cuando se activa un nodo nexo se activan todos los nodos contenedores, los nodos activadores de nexos. También se activan (una vez satisfecha la restricción temporal) los nodos contenedores con conexión sincro (paneles de la misma ventana), y los nodos nexo con enlace sincro (otra ventana distinta). Además todos los nodos contenedores muestran su contenido por defecto, con o sin restricciones temporales. Por tanto lo que se almacena en activos y mostrar es la información sobre cuando se debe activar el nodo en concreto. Nótese que para que esta semántica funcione, es necesario que el componente encargado de reproducir los contenidos, debe ser capaz de llevar cuenta de la información temporal. Hemos obviado este componente, ya que lo único que hace es reproducir un contenido si no tiene información temporal, y esperar a que se cumpla la restricción correspondiente para reproducirlo. Es decir, realmente faltaría una función reproducir,

$$\text{reproducir}: (C \cup (C \times \mathbb{R} \times N_x)) \rightarrow \Omega$$
$$\text{reproducir}(c) = \omega(c)$$
$$\text{reproducir}(c, t, x_i) = \omega(c), \text{ si } t = \text{tiempo}(x_i)$$

Donde lo único que hace esta función es reproducir el contenido ($\omega(c)$) si no hay restricciones temporales, y reproducirlo acorde a estas restricciones temporales si las hay ($\omega(c)$, si $t = \text{tiempo}(x_i)$, supuesto que la función $\text{tiempo}(x_i)$ informe de la cantidad de unidades de tiempo transcurridas desde la activación del nodo nexo x_i).Como toda la información que necesita esta función se encuentra en el conjunto $Mostrar_n$, obviamos su inclusión. Además dicha función debería tener acceso (incluirse en su dominio) a los nodos contenedores, y a la función de asignación de contenidos para garantizar que puede llevar cuenta de las restricciones temporales impuestas.

Ahora debemos indicar que sucede cuando se activa un enlace de contenidos. Definimos la función de activación de enlaces f,

$$f: ((C_{ns} \times A) \cup \{\perp\}) \times N \times \wp(V) \times \wp(M) \rightarrow \wp(V) \times \wp(M)$$

Donde el símbolo $\perp$ se utiliza para los enlaces que no tienen origen en enlaces de contenido (es decir los sincro entre nodos nexo, y los de activador de nexo). Veamos que sucede con f según la naturaleza del nodo en que se origine el enlace, y el nodo destino del enlace.

- El origen se produce en un nodo de contenidos, y el destino es otro nodo de contenidos, es decir,

  $$f((o, a), c_i, Activos_n, Mostrar_n) = (Activos_{n+1}, Mostrar_{n+1})$$

y $\exists k \forall m(m \in ne(o, a) \Rightarrow \exists j ((c_i, c_j, enlace) \in A \wedge enlAct(o, a, m) \in l(c_i, c_j, enlace) \wedge (x_k, c_j, conexión) \in A)) \wedge (c_i, o) \in Mostrar_n$

Lo que exige esta condición es que haya un nodo nexo que contenga a tantos nodos contenedores como destinos del enlace n-ario haya, y que además los enlaces con origen en el ancla seleccionada estén canalizados por tuberías que desemboquen en dichos contenedores. También exige que realmente estemos mostrando el contenido del cual seleccionamos el enlace (nótese que esto garantiza que el nodo del cual parte el enlace esté activo). Veamos que sucede si el nodo destino pertenece o no al mismo nodo nexo, es decir,

- Si $((x_r, c_i, conexión) \in A \wedge r \neq k)$, es decir, tenemos una situación como la descrita en la Figura A.6.

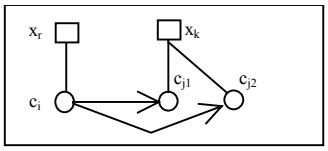

Figura A.6 Esquema navegacional posible

$Activos_{n+1} = \prod_1 a(x_k)$

$Mostrar_{n+1} =$
$\{ (c_j, conAcc(o, a, m)) \mid m \in ne(o, a) \wedge enlAct(o, a, m) \in l(c_i, c_j, enlace) \}$
$\cup \{ (c_j, d_1(c_j)) \mid (x_k, c_j, conexión) \in A \wedge \forall m \in ne(o, a)(enlAct(o, a, m) \notin l(c_i, c_j, enlace)) \}$
$\cup \{ (c_j, d_1(c_j), t) \mid (x_k, c_j, conexiónS, t) \in A \}$

Es decir, se activa el nodo nexo que contiene al nodo contenedor destino de la tubería, y todos los nodos muestran sus contenidos por defecto, salvo los nodos contenedor que muestran el destino de los enlaces navegacionales a través de la función conAcc. En el caso enlaces estáticos devuelve el extremo del enlace. En caso de enlaces dinámicos devuelve el contenido generado. Esto puede hacerse así por la definición de l, teniendo en cuenta a la función d. Al desestimar $Activos_n$ y $Mostrar_n$ "eliminamos" la ventana de la que partió el enlace. Nótese que aunque en $Mostrar_n$ obviamos la información sobre el ancla destino, esta se incluye en $\prod_2 r(o, a, m)$. Los nodos con conexión sincro guardan su hora de activación.

- Si $(x_k, c_i, conexión) \in A$, es decir, tenemos una situación como la descrita en la Figura A.7.

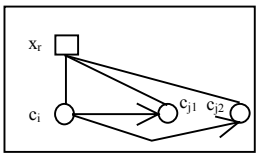

Figura A.7 Esquema navegacional posible

$Activos_{n+1} = Activos_n$

$Mostrar_{n+1} = Mostrar_n \setminus$
$\{(c_j, x) \mid (c_j, x) \in Mostrar_n \wedge m \in ne(o, a) \wedge enlAct(o, a, m) \in l(c_i, c_j, enlace)\}$
$\cup \{(c_j, conAcc(a, o, m)) \mid m \in ne(o, a) \wedge enlAct(o, a, m) \in l(c_i, c_j, enlace)\}$

Es decir, se activa cada nodo contenedor y se muestra el contenido seleccionado por cada enlace (bien estático o dinámico) dejando de mostrar el contenido previo.

- El origen se produce en un nodo nexo, y por tanto es un enlace sincro. Nótese que en este caso no tenemos enlace de contenidos y usamos ⊥. Es decir, estamos en una situación descrita como en la Figura A.9.

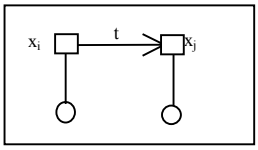

Figura A.8 Esquema navegacional posible

$$f(\perp, x_i, Activos_n, Mostrar_n) = a(x_j)$$
$$\text{si } ((x_j, t) \in Activos_n \wedge t = tiempo(x_i))$$

Donde la función tiempo(x_i) devuelve el tiempo transcurrido desde la activación del nodo nexo x_i.
Nótese que es necesario que solo parta un enlace sincro del nodo origen, y que nos limitamos a activar el nodo destino cuando han transcurrido t segundos desde la activación de x_i.

- El origen se produce en un activador de nexo. Nótese que en este caso no tenemos enlace de contenidos y usamos ⊥. Es decir, estamos en una situación como la descrita en la Figura A.9.

$$f(\perp, a_i, Activos_n, Mostrar_n) = a(x_j) \text{ si } \exists\, j\ ((a_i, x_j, enlace) \in A)$$

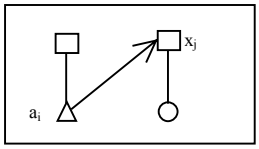

Figura A.9 Esquema navegacional posible

En este caso simplemente activamos el nodo destino. Nótese que por como hemos definido A, los activadores de nexos solo pueden enlazar con un destino.

- En otro caso, f se comporta como la identidad.

En el caso de la semántica presentacional se procede de la misma forma.

A.2.5 Ejemplo

Ejemplo estático

Supongamos que tenemos un índice general que da acceso a dos índices sobre dos tópicos distintos y al primer contenido de cada tópico. Es decir, una situación como la descrita en la Figura A.10.

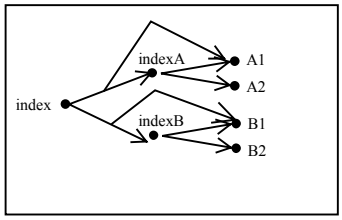

Figura A.10 Grafo de contenidos de la aplicación.

En esta ocasión los enlaces binarios quedan representados por flechas con un mismo origen. La versión conjuntista de la Figura 11 viene proporcionada por los siguientes conjuntos.

C^A = { index, indexA, indexB, A1, A2, B1, B2 }

E^A = { (index, aIAA1, 1, indexA, total), (index, aIAA1, 2, A1, total),
(index, aIBB1, 1, indexB, total), (index, aIBB1, 2, B1, total),
(indexA, aA1, 1, A1, total), (indexA, aA2, 1, A2, total),
(indexB, aB1, 1, B1, total), (indexB, aB2, 1, B2, total) }

Supongamos que deseamos presentar toda la información en una sola ventana. En un panel el índice general, en otro los subíndices y en el tercero los contenidos. La Figura A.11 describe dicho esquema.

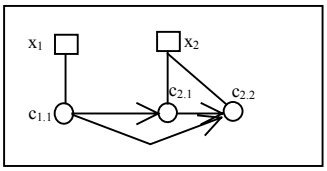

Figura A.11 Esquema navegacional

El esquema navegacional está representado por los siguientes conjuntos.

N = { x1, c1.1, x2, c2.1, c2.2 }

A = { (x1, c1.1, conexión), (x2, c2.1, conexión), (x2, c2.2, conexión),
(c1.1, c2.1, enlace), (c1.1, c2.2, enlace), (c2.1, c2.2, enlace) }

Formalmente, la asignación de contenidos y enlaces viene dada por las siguientes funciones.

d = { (**c1.1**, index, ∅), (**c2.1**, null, {indexA, indexB}), (**c2.2**, null, {A1, A2, B1, B2}) }

l = { (**c1.1, c2.1, enlace**, { (index, aIAA1, 1, indexA, total), (index, aIBB1, 1, indexB, total)}),
(**c1.1, c2.2, enlace**, { (index, aIAA1, 2, A1, total), (index, aIBB1, 2, B1, total)}),
(**c2.1, c2.2, enlace**, { (indexA, aA1, 1, A1, total), (indexA, aA2, 1, A2, total)
(indexB, aB1, 1, B1, total), (indexB, aB2, 1, B2, total) }) }

Ahora, supuesto que tenemos activado el nodo contenedor x1, si activamos un enlace binario obtenemos los resultados deseados. En efecto:

f((index, aIAA1), c1.1, {x1, c1.1}, {(c1.1, index)})

$\exists 2\ \forall m$ ((m ∈ ne(index, aIAA1) ⇒ ∃j((c1.1, cj, enlace)∈ A ∧ enlAct(index, aIAA1, m)∈ l(c1.1, cj, enlace)
∧ (x2, cj, conexión)∈ A))
∧ (c1.1, index)∈ Mostrar1

En efecto, como ne(index, aIAA1) = { 1, 2 }, los pares (m, j) que hacen cierta la expresión anterior son (1, 2.1) y (2, 2.2).

Como estamos en la situación en la que: $\exists 1$((x1, c1.1, conexión) ∈ A ∧ 1≠2)

Tenemos que:

Activos2 = {x2, c2.1, c2.2}
Mostrar2 = {(c_j, conAcc(index, aAA1, m)) | m∈ ne(index, aAA1)
∧ enlAct(index, aAA1, m) ∈ l(c1.1, c_j, enlace)}
= {(c2.1, indexA), (c2.2, A1) }

Que es justo lo que queríamos, tal y como indica la Figura A.12.

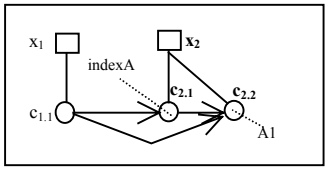

Figura A.12 Enlace binario activado

Si hubiésemos deseado un esquema navegacional en el que todo se mostrase en una única ventana, es decir, según la Figura A.13.

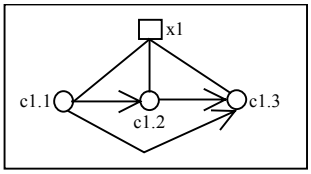

Figura A.13 Esquema navegacional alternativo

Dicha figura en versión conjuntista viene representada por los siguientes conjuntos.

N = {x1, c1.1, c1.2, c1.3 }

A = { (x1, c1.1, conexión), (x1, c1.2, conexión), (x1, c1.3, conexión),
(c1.1, c1.2, enlace), (c1.1, c1.3, enlace), (c1.2, c1.3, enlace) }

La asignación de nodos y enlaces viene dada por los siguientes conjuntos.

d = { (**c1.1**, index, ∅), (**c1.2**, null, {indexA, indexB}), (**c1.3**, null, {A1, A2, B1, B2}) }

l = { (**c1.1, c1.2, enlace**, { (index, aIAA1, 1, indexA, total), (index, aIBB1, 1, indexB, total)}),
(**c1.1, c1.3, enlace**, { (index, aIAA1, 2, A1, total), (index, aIBB1, 2, B1, total)}),
(**c1.2, c1.3, enlace**, { (indexA, aA1, 1, A1, total), (indexA, aA2, 1, A2, total)
(indexB, aB1, 1, B1, total), (indexB, aB2, 1, B2, total) }) }

Ahora, supuesto que tenemos activado el nodo contenedor x1, si activamos un enlace binario obtenemos los resultados deseados. En efecto:

f((index, aIAA1), c1.1, {x1, c1.1, c1.2, c1.3}, {(c1.1, index), (c1.2, null), (c1.3, null)})

$\exists 1\ \forall m$ ((m ∈ ne(index, aIAA1) $\Rightarrow$ $\exists$j((c1.1, cj, enlace)∈ A $\wedge$ enlAct(index, aIAA1, m)∈ l(c1.1, cj, enlace)
$\wedge$ (x1, cj, conexión)∈ A))
$\wedge$ (c1.1, index)∈ Mostrar1

En efecto, como ne(index, aIAA1) = { 1, 2 } pares (m, j) que hacen cierta la expresión anterior son (1, 1.2) y (2, 1.3).

Como estamos en la situación en la que: (x1, c1.1, conexión) ∈ A

Tenemos que:

Activos2 = Activos1
Mostrar2 = Mostrar1
\ {(c_j, x) | (c_j, x) ∈ Mostrar1 $\wedge$ m∈ ne(index, aAA1) $\wedge$ enlAct(index, aAA1, m) ∈ l(c1.1, c_j, enlace)}
$\cup$ {(c_j, enlAct(index, aIAA1, m)) | m∈ ne(index, aAA1) $\wedge$ enlAct(index, aAA1, m) ∈ l(c1.1, c_j, enlace)}
= {(c1.1, index), (c1.2, indexA), (c1.3, A1) }

Que es justo lo que queríamos, tal y como indica la Figura A.14.

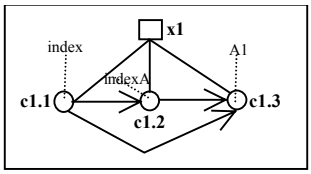

Figura A.14 Enlace binario activado

Ejemplo dinámico

Supongamos ahora un ejemplo dinámico como el propuesto por la Figura A.15.

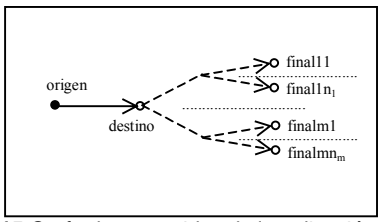

Figura A.15 Grafo de contenidos de la aplicación no permitido

En principio esos contenidos no pueden tener asignado un esquema navegacional en el modelo, ya que para mostrar los contenidos destino del enlace n-ario (por ejemplo, final11, ..., final1n_1) serían necesarios un número no determinado (en este caso n_1) de nodos contenedor para asignarlos. Nótese que este supuesto no está permitido por la definición de la función de asignación de enlaces l. Por tanto, el número de contenidos del enlace n-ario debe conocerse de antemano. En particular tomemos dos. Este problema no existía con los enlaces unarios, ya que era posible asignar todos los contenidos generados, así como sus enlaces a un solo nodo y tubería a través de las funciones conectados y enlazados. Pues bien, limitemos los contenidos a los mostrados por la Figura A.16.

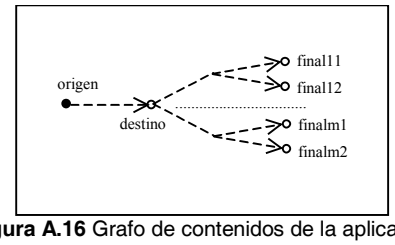

Figura A.16 Grafo de contenidos de la aplicación.

En este caso la generación del primer contenido dinámico viene representada en la Tabla A.2.

(origen, aD, 1)	(destino, total, {(aF11F12,1), (aF11F12, 2), ...,(aFM1FM2, 1), (aFM1FM2, 2)}, {total})

Tabla A.2 Primera aplicación de la función g

Es decir, tenemos que.

cg(origen, aD, 1) = destino
eg(origen, aD, 1) = (origen, aD, 1, destino, total)
ca(cg(origen, aD, 1)) = {(destino, aF11F12, 1), ..., (destino, aFM1FM2, 2)}

La segunda aplicación de la función de generación de contenidos viene representada en la Tabla A.3.

(destino, aF11F12, 1)	(final11, total, ∅, {total})
(destino, aF11F12, 2)	(final12, total, ∅, {total})
.....	
(destino, aFM1FM2, 1)	(finalm1, total, ∅, {total})
(destino, aFM1FM2, 2)	(finalm2, total, ∅, {total})

Tabla A.3 Segunda aplicación de la función g

Si consideramos un esquema navegacional de dos ventanas, donde la primera contiene al contenido estático, y en la segunda, en dos paneles distintos, los contenidos generados dinámicamente, tenemos un esquema como el descrito por la Figura A.17.

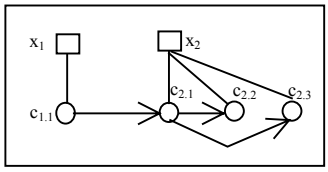

Figura A.17 Esquema navegacional

La asignación de contenidos viene dada por la siguiente función.

d = { (**c1.1**, origen, ∅), (**c2.1**, null, cg(origen, aD, 1)), (**c2.2**, null, FINAL1), (**c2.3**, null, FINAL2) }

Siendo,

destino = cg(origen, aD, 1)
FINAL1 = { cg(destino, a, 1) | (a, 1)∈ AGO ∧ AGO = Π_3g(origen, aD, 1) }
FINAL2 = { cg(destino, a, 2) | (a, 2)∈ AGO ∧ AGO = Π_3g(origen, aD, 1) }

Como AGO = Π_3g(origen, aD, 1) = {(aF11F12,1), (aF11F12, 2), ...,(aFM1FM2, 1), (aFM1FM2, 2)} tenemos que:

FINAL1 = {final11, ..., finalm1}
FINAL2 = {final12, ..., finalm2}

La canalización de enlaces viene dada por la siguiente función.

l = { (**c1.1, c2.1, enlace**, eg(origen, aD, 1)),
(**c2.1, c2.2, enlace**, ENLACES1), (**c2.1, c2.3, enlace**, ENLACES2) }

Siendo,

ENLACES1 = { eg(destino, a, 1) | (a, 1)∈ AGO ∧ AGO = Π_3g(origen, aD, 1) }
ENLACES2 = { eg(destino, a, 2) | (a, 2)∈ AGO ∧ AGO = Π_3g(origen, aD, 1) }

Como AGO = Π_3g(origen, aD, 1) = {(aF11F12,1), (aF11F12, 2), ...,(aFM1FM2, 1), (aFM1FM2, 2)} tenemos que:

ENLACES1 = { (destino, aF11F12, 1, final11, total), ..., (destino, aFM1FM2, 1, finalm1, total) }
ENLACES2 = { (destino, aF11F12, 2, final12, total), ..., (destino, aFM1FM2, 2, finalm2, total) }

Ahora, supuesto que tenemos activado el nodo contenedor x1, si activamos el enlace unario que genera el primer contenido dinámico tenemos lo siguiente.

f((origen, aD), c1.1, {x1, c1.1}, {(c1.1, origen)})

$\exists 2\ \forall m$ ((m ∈ ne(origen, aD) ⇒ $\exists$j((c1.1, cj, enlace)∈ A ∧ enlAct(origen, aD, m)∈ l(c1.1, cj, enlace)
∧ (x2, cj, conexión)∈ A))
∧ (c1.1, origen)∈ Mostrar1

En efecto, como ne(origen, aD) = { 1 }, el (m, j) que hacen cierta la expresión anterior es (1, 2.1).

Como estamos en la situación en la que: $\exists$1((x1, c1.1, conexión) ∈ A ∧ 1≠2)

Tenemos que:

Activos2 = {x2, c2.1, c2.2, c2.3}
Mostrar2 = {(c_j, conAcc(origen, aD, m)) | m∈ ne(origen, aD) ∧ enlAct(origen, aD, m) ∈ l(c1.1, c_j, enlace)}
∪ {(c_k, d_1(c_k)) | (x2, c_k, conexión) ∈ A ∧ ∧ ∀m∈ ne(origen ,aD)(enlAct(origen, aD, m) ∉ l(c_i, c_j, enlace))
}

$\cup$ {(c_k, $d_1(c_k)$, t)| (x2, c_k, conexiónS, t)$\in$ A $\wedge$ k$\neq$j }
= {(c2.1, cg(origen, aD, 1) } $\cup$ {(c2.2, null), (c2.3, null)}
= {(c2.1, destino), (c2.2, null), (c2.3, null) }

Tal y como muestra la Figura A.18.

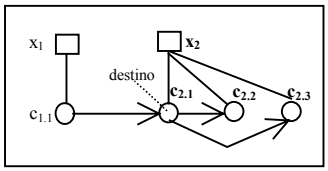

Figura A.18 Activación del enlace unario

Si ahora seleccionamos un enlace binario.

f((destino, aF11F12), c2.1, Activos2, Mostrar2)

$\exists 2\ \forall m$ (m $\in$ ne(destino, aF11F12) $\Rightarrow \exists j$((c2.1, cj, enlace)$\in$ A $\wedge$ enlAct(destino, aF11F12, m)$\in$ l(c2.1, cj, enlace)
$\wedge$ (x2, cj, conexión)$\in$ A))
$\wedge$ (c2.1, destino)$\in$ Mostrar2

Como ne(destino, aF11F12) = { 1, 2 }, los (m, j) que hacen cierta la expresión anterior son (1, 2.2) y (2, 2.3), ya que,

enlAct(destino, AF11F12, 1) $\in$ ENLACES1$\in$l (c2.1, c2.2, enlaces)
enlAct(destino, AF11F12, 2) $\in$ ENLACES2 $\in$ l(c2.1, c2.3, enlaces)

Como estamos en la situación en la que (x2, c2.2, enlace)$\in$ A, tenemos que:

Activos3 = Activos2
Mostrar3 = Mostrar2
\ {(cj, x) | (cj, x) $\in$ Mostrar2 $\wedge$ m$\in$ne(destino, aF11F12) $\wedge$ enlAct(destino, aF11F12, m) $\in$ l(c2.1, cj, enlace) }
$\cup$ {(cj, enlAct(destino, aF11F12, m)) | m$\in$ne(destino, aF11F12) $\wedge$ enlAct(destino, aF11F12, m) $\in$ l(c2.1, cj, enlace) }
= {(c2.1, destino), (c2.2, final11), (c2.3, final12) }

Que es el resultado que deseabamos, tal y como muestra la Figura A.19.

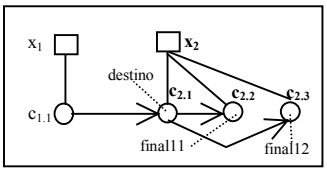

Figura A.19 Activación del enlace binario

Apéndice B. GAP

B.1 Introducción

En este apéndice se incluye el diseño del Generador Automático de Prototipos, GAP, que implementa al modelo Pipe. GAP es una aplicación Java 2 (aunque hay muchas, [Niemeyer 00] es una valiosa referencia) que se beneficia de las capacidades proporcionadas por este lenguaje de programación (en particular por la API Swing, de la que [Walrath 99] representa una de las más completas referencias), y de las APIs de estándares relacionados con XML para él. En particular GAP se ha desarrollado sobre JDK 1.3.0 [Sun Java 2] utilizando el entorno CodeWarrior 5 (sobre Windows 98, patch para JDK 1.3 incluido) básicamente, aunque los últimos retoques fueron dados con el entorno CodeWarrior 6 (sobre Windows XP). Además se han utilizado las APIs XML4J 3.1.1 (o Xerces) [IBM Xerces] y Xalan 2.0.0 (actual Lotus XSL) [IBM Xalan]. La primera proporciona el soporte DOM necesario. La segunda proporciona el soporte XPath y XSLT necesario. El uso de ambas APIs junto a la potencia de Java han simplificado notablemente el desarrollo de GAP. Respecto al modelo de proceso se optó por un modelo incremental, entregando versiones funcionalmente completas en cada incremento. Aunque se partió de la quinta interacción de un generador de prototipos anterior (sobre la base de una versión previa de Pipe no presentada en esta tesis) se hicieron un total de cinco incrementos. En el primero se incluyó el nuevo vocabulario respecto a la versión previa, además de hacer otra serie de adaptaciones. En el segundo se dio soporte a las capacidades del nuevo Pipe. En el tercero se introdujo el control de excepciones para dar un ligero soporte al usuario. En el cuarto se introdujeron procesos subordinados. En el quinto se mejoro el código sin añadir ninguna funcionalidad nueva. El esfuerzo total de desarrollo (las versiones previas más estas cinco versiones) no excedió la cantidad de una persona-mes.

En cuanto al código resultante, el cual se puede encontrar en el apartado B1.3 es importante resaltar la sencillez del mismo, resultante de la integración de las APIs Swing, DOM, y XSLT. De no haber contado con estas APIs, las líneas de código (y por extensión el esfuerzo de desarrollo) habrían aumentado sobremanera.

Veamos brevemente *que* hace GAP. GAP genera una aplicación hipermedia a partir de:

- Los contenidos enlazados de la misma descritos mediante un documento XML, el documento de contenidos. Los contenidos en formato no XML (imágenes y procesos) son referenciados por elementos del documento.
- El esquema presentacional de la aplicación descrito mediante un documento XML, que hace referencia a los contenidos mediante expresiones XPath, el documento de la aplicación.
- Clases Java que implementan a los procesos subordinados.

Veamos ahora *cómo* lo hace:

- La aplicación generada es un array de ventanas, siendo una ventana un `JFrame`.
- Cada ventana está formada por un array de paneles y de activadores de ventanas. Los paneles son `JPanel` y los activadores de ventana son `JButton`.
- Ventanas, paneles y activadores se inicializan a partir del tratamiento del documento de la aplicación.
- En este tratamiento inducido por el documento de la aplicación se generan los contenidos no subordinados de la aplicación como páginas HTML a partir del documento de contenidos utilizando una transformación XSLT estándar. Los contenidos en formato no XML (imágenes y procesos) son referenciados por elementos del documento.
- Dichas páginas HTML se cargan en un `JEditorPane` que contiene cada panel.
- Las clases (procesos subordinados) referenciados en el documento de contenidos se cargan en un `JPanel` que también contiene el panel.
- El único evento al que tiene que responder la aplicación es la selección de un hiperenlace en una página HTML, lo cual se soluciona mediante una implementación del interfaz `HyperlinkListener`. Las clases Java cargadas responden por ellas mismas a los eventos generados en su panel.

La Figura B.1 describe esta aproximación.

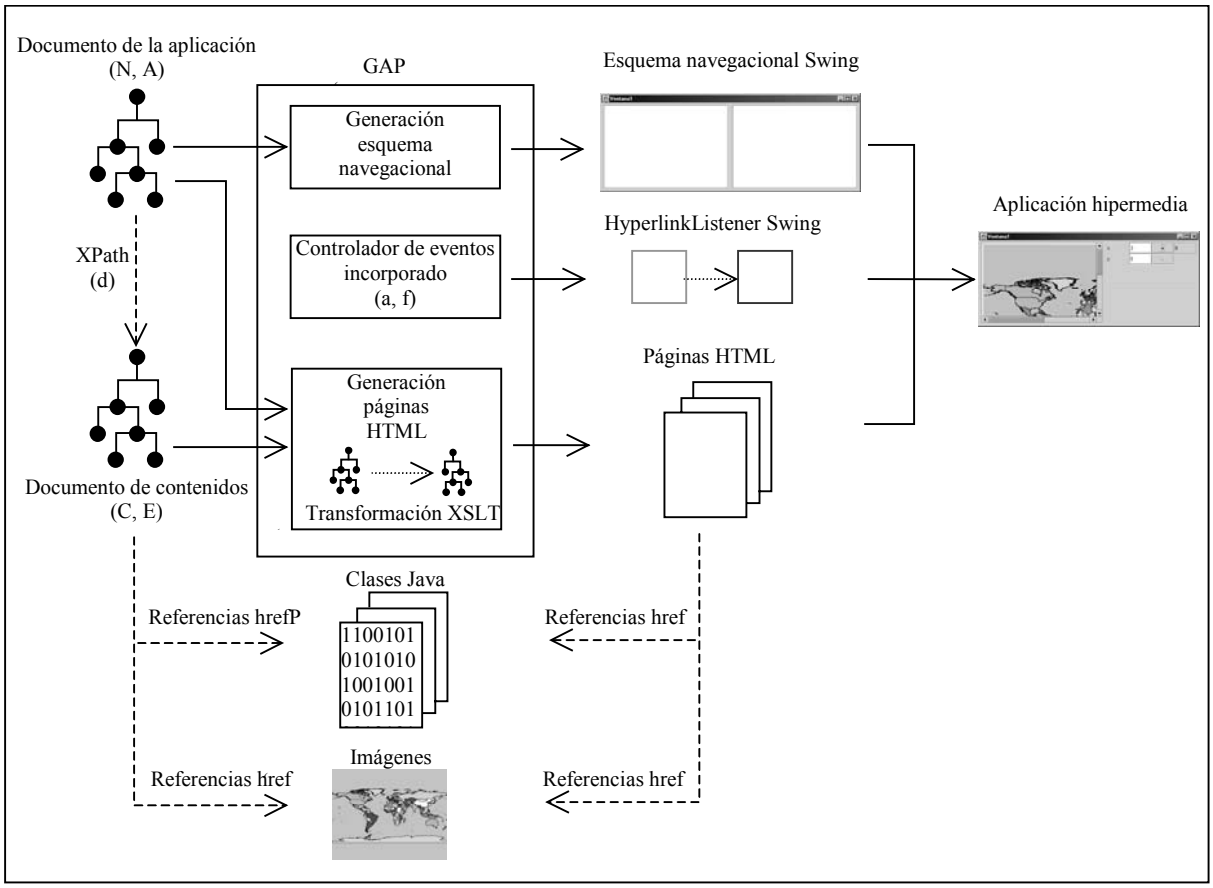

Figura B.1 Esquema de funcionamiento GAP

A continuación veremos una descripción en detalle del diseño de la aplicación.

B.2 Diseño

Arquitectura

Como ya hemos comentado en la introducción, la implementación esta basada en la API Swing de Java. La arquitectura software de la aplicación puede verse en la Figura B.2.

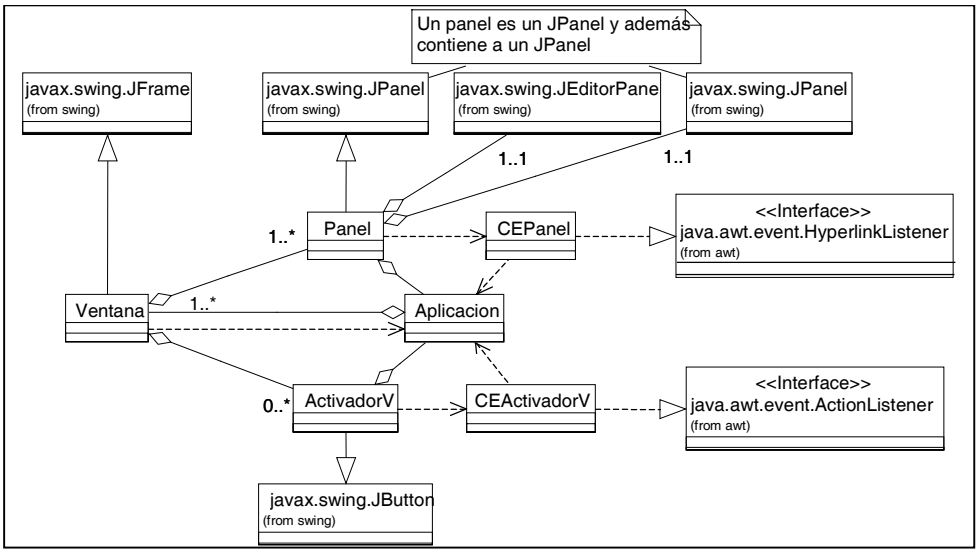

Figura B.2 Arquitectura GAP

La única actividad que debe llevar a cabo GAP es la de inicializar la aplicación hipermedia. La Figura B.3 describe esta actividad, así como las actividades que la componen.

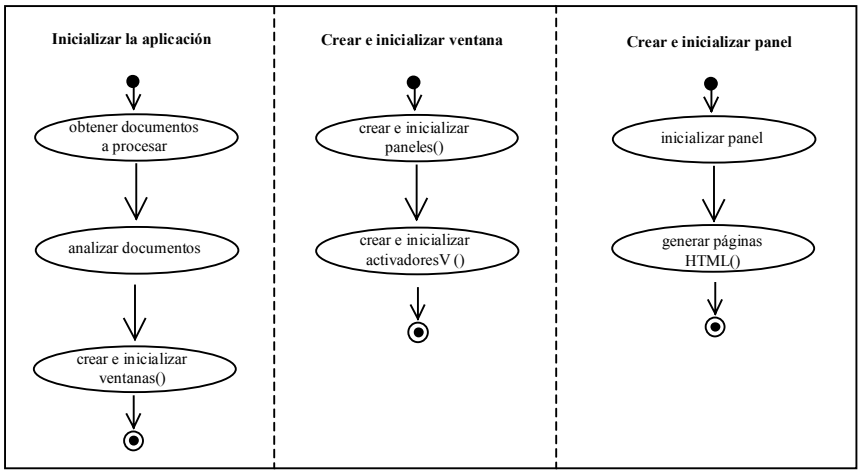

Figura B.3 Actividades principales en GAP. Hemos obviado el tratamiento de errores para aligerar los diagramas

A continuación veremos las cuatro clases fundamentales de la arquitectura GAP.

Aplicación

Esta clase se encarga de obtener los nombres de archivos XML, de analizarlos utilizando un objeto Xerces `DOMParser` y de crear e inicializar las ventanas de la aplicación hipermedia. Además proporciona los métodos para que los paneles y activadores de ventana puedan responder convenientemente a los eventos que se producen. La Figura B.4 muestra a esta clase.

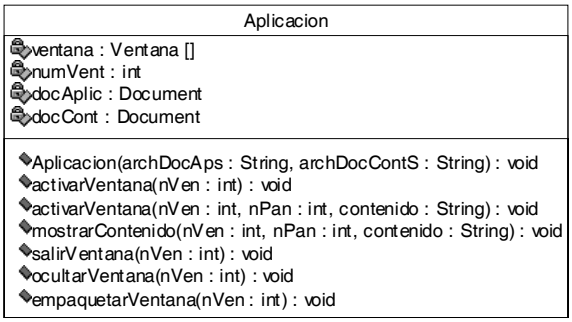

Figura B.4 La clase Aplicacion

Ventana

Una ventana básicamente se encarga de inicializar sus paneles y activadores de ventana. También proporciona las funciones necesarias para responder a su activación, o para mostrar contenidos. La Figura B.5 describe a esta clase.

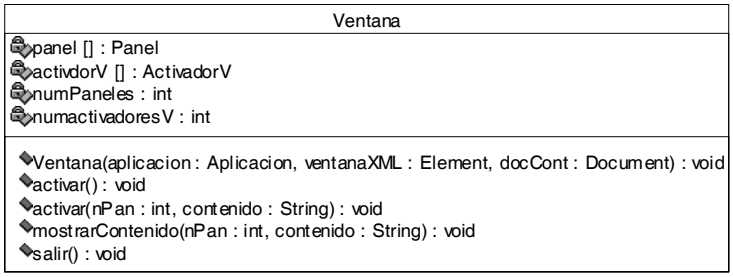

Figura B.5 La clase Ventana

Nótese como el argumento `aplicacion` pone de manifiesto la dependencia entre la clase `Ventana` y la clase `Aplicacion`.

Panel

Esta clase es la más compleja dentro de GAP, ya que además de inicializar los paneles es la encargada de generar los contenidos de la aplicación. La Figura B.6 describe a esta clase.

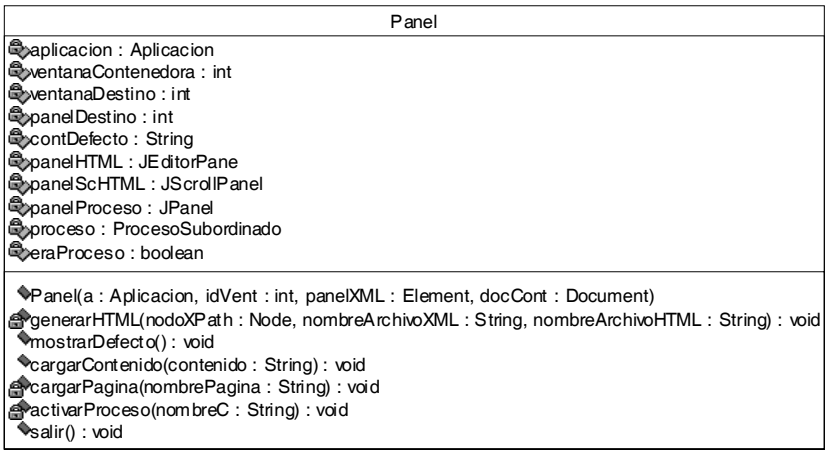

Figura B.6 La clase Panel

Aunque no va a contener simultáneamente a una página HTML y a un proceso subordinado no cabe especializarlo, ya que durante la vida de los objetos de esta clase es posible que contengan tanto páginas HTML como procesos subordinados. Un proceso subordinado no es más que una clase abstracta que se encarga de garantizar la presencia de un panel para construir su interfaz de usuario (el `panelProceso` de `Panel`), así como la presencia de una función para activarlo y otra para detenerlo. La Figura B.6 describe a la clase `ProcesoSubordinado`.

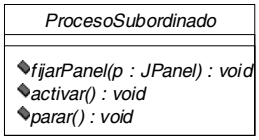

Figura B.7 La clase ProcesoSubordinado

Hemos optado por considerarlo interfaz en vez de clase abstracta para condicionar lo menos posible el código de la clase que lo implemente (no olvidemos que en Java no hay herencia múltiple). En cualquier caso, el cambio a clase abstracta conllevaría cambios mínimos en GAP.

Por otro lado la interacción que se produce entre los objetos que inicializan el panel y generan las páginas HTML puede verse en el diagrama de secuencia de la Figura B.8. Nótese que cuando el contenido es un proceso subordinado no se genera ningún tipo de contenido. La razón se debe a que estos procesos son nombres de clases Java especificadas como contenido por defecto en el documento de la aplicación. Cuando una página se activa simplemente debe cargarse la clase especificada.

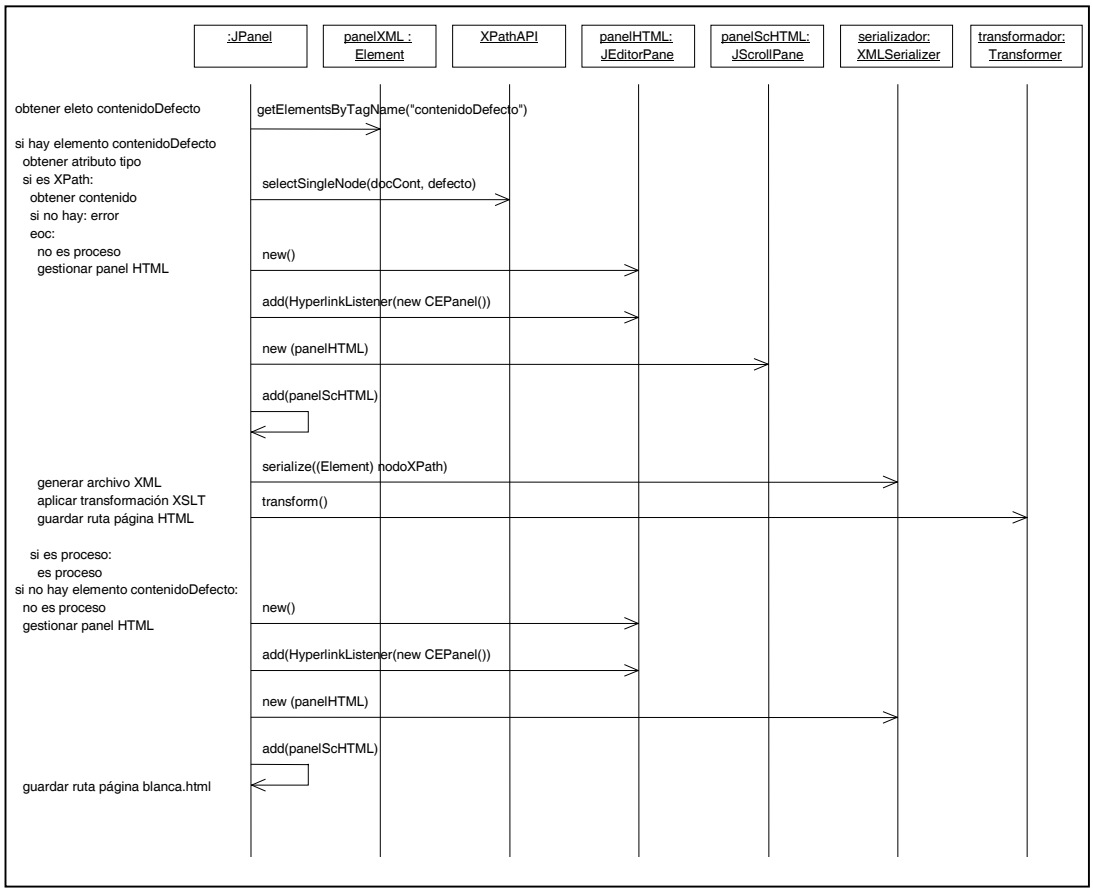

Figura B.8 Interacción producida durante el procesamiento de la etiqueta contenidoDefecto

El tratamiento de un elemento contenido es similar, pero obviando cualquier gestión de la interfaz. Nótese que por tanto en ningún momento la asignación de contenido a un panel limita los contenidos que pueden aparecer en él. Simplemente son un medio de aplicarles la transformación XSLT para generar las páginas HTML. En el caso de los procesos referenciados en el documento de contenidos mediante un atributo de nombre `hrefP`, la transformación XSLT lo traduce a un atributo `href`. Cuando se selecciona el enlace cuyo destino es el proceso subordinado se detecta (el nombre de la clase no lleva extensión) y se activa el proceso. Por tanto, tal y como comentamos en el Capítulo 6, el especificar un contenido (no por defecto) como proceso subordinado tiene una misión meramente declarativa, ya que no induce ningún procesamiento en GAP.

Una cuestión interesante en la generación de las páginas HTML es que nombre asignar a los contenidos. A los contenidos por defecto se les asigna el nombre del panel que los contiene. A los contenidos referenciados en el documento de contenidos se les asigna como nombre su identificador. De esta forma cuando son referenciados pueden ser convenientemente localizados. Esta es la razón por la cual los elementos localizados por una expresión XPath de una etiqueta `contenido` o los subelementos de una etiqueta `contenidoGrupo` deben tener un atributo `id` de tipo `ID`. Nótese además que este tipo de tratamiento es el que obliga a incluir en contenido un `contenidoDefecto` que va a ser referenciado. Una posible solución sería el generar la página HTML con el nombre del `id` del contenido por defecto en el caso de que este tuviese, por supuesto junto a la página cuyo nombre es el del panel correspondiente.

La interacción que se produce al cargar un proceso subordinado en Pipe puede verse en el diagrama de secuencia de la Figura B.9.

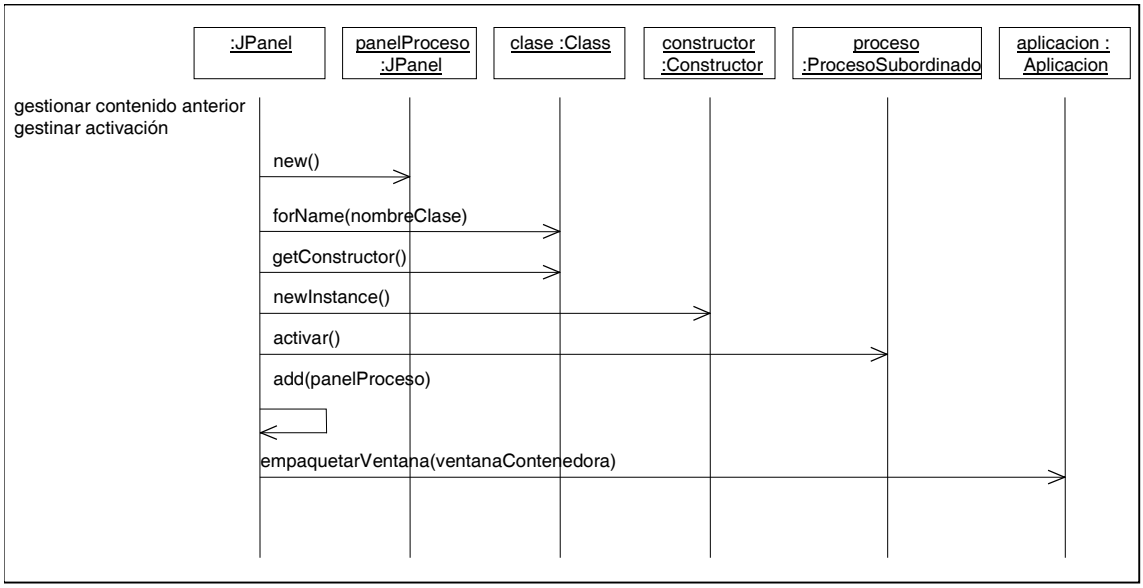

Figura B.9 Interacción producida durante la activacion de un proceso subordinado

Otra cuestión interesante son los cambios que sufre un panel cuando cambia de un contenido HTML a un contenido proceso. Lo que se debe hacer en cada caso, y la forma de detectarlo quedan recogidas en las Tabla B.1 y en la Tabla B.2.

Enlace misma ventana

(ahora, antes)	*HTML*	*proceso*
HTML	cargar página	parar proceso quitar panel proceso poner panel HTML cargar página
proceso	quitar panel HTML poner panel proceso activar proceso	parar proceso quitar panel proceso poner panel proceso activar proceso

Enlace otra ventana

(ahora, antes)	*HTML*	*proceso*
HTML	cargar página	poner panel HTML cargar página
proceso	quitar panel HTML poner panel proceso activar proceso	poner panel proceso activar proceso

Tabla B.1 Distintas opciones a la hora de activar un panel. El tipo de contenido (ahora) se detecta con la función esArchivo(). El tipo de contenido anterior (antes) se detecta con el campo eraProceso. Si el enlace se produce en la misma ventana se detecta preguntando si (proceso != null).

La activación de un hiperenlace de contenidos en el esquema navegacional esta capturada por la clase `CEPanel` que implementa al interfaz `HyperlinkListener`. La interacción que se produce en su método `hyperlinkUpdate()` puede verse en la Figura B.10.

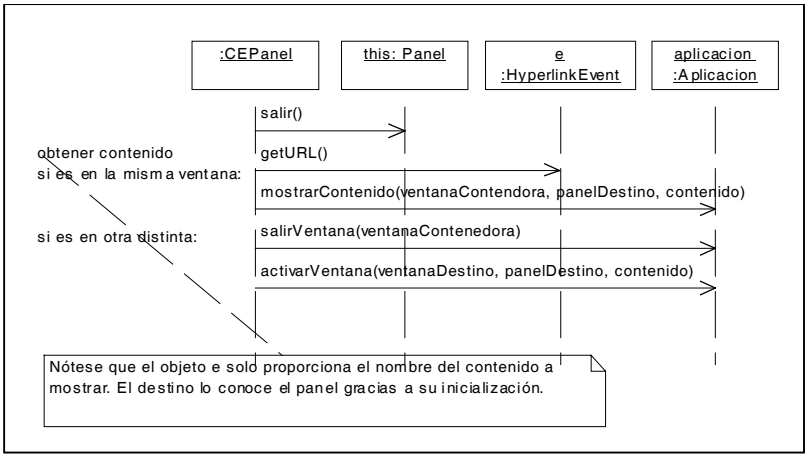

Figura B.10 Interacción producida durante la selección de un hiperenlace

ActivadorV

Esta clase es extremadamente sencilla, pues no es más que un `JButton` que guarda el código de ventana que debe activar. La Figura B.11 muestra a esta clase.

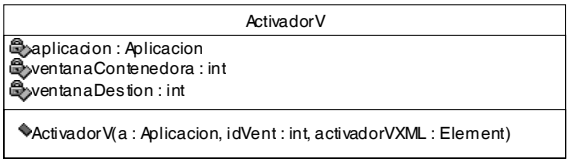

Figura B.11 La clase ActivadorV

Lo más interesante de esta clase radica en la clase interna `CEActivadorV` que implementa al interfaz `ActionListener`. La interacción que se produce en el método `actionPerformed()` puede verse en la Figura B.12.

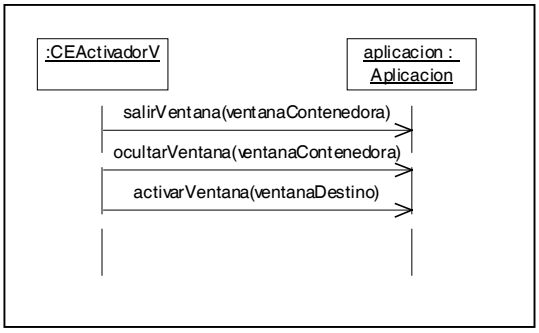

Figura B.12 Interacción al activar un activador de ventana

Otras clases

GAP utiliza un par de clases más auxiliares. En particular la clase `DPErrorHandler` y la clase `Error`. La primera es una implementación del `ErrorHandler` de SAX a través de una especialización del `HandlerBase`. Aunque en SAX 2.0 ya no se sigue esta aproximación, debido a que es una clase que solamente se utiliza para validar los documentos hemos optado por proporcionar su implementación, en aras de una mayor sencillez. Por otro lado, la clase `Error` no es más que una clase con un único método `static` (`mensaje()`) que muestra un panel titulado `Error` en el cual se incluye un mensaje proporcionado por el usuario de la clase.

Restricciones de uso

Existen una serie de restricciones, o normas de uso para la creación del documento de contenidos y de la aplicación debidas a la implementación del generador de prototipos GAP proporcionada. Estas son:

- Restricciones en el documento de la aplicación:
- Las ventanas deben numerarse de forma consecutiva y con la nomenclatura v_i.
- Los paneles deben numerarse de forma consecutiva con la nomenclatura $p_{i.j}$ donde la `i` denota la ventana a la que pertenece y la `j` el número de panel.
- En los activadores de ventana no es necesario utilizar una nomenclatura especial, pero por consistencia se utilizara $av_{i.j}$.
- La ventana inicial es la v_1.
- Si se tiene un único panel para mostrar los contenidos, y se va a referenciar el contenido por defecto del panel desde otro contenido es necesario incluir al contenido por defecto del panel como `contenido`. Esto se debe a que cuando se genera un contenido por defecto a la página HTML generada se le asigna el nombre del panel al cual se le asigna, pero si es referenciada desde los contenidos, entonces se referencia a una página HTML cuyo nombre es su identificador. Para generar esta página es necesario asignarle como `contenido`.

- Restricciones en el documento de contenidos:
  - Los contenidos referenciados mediante etiquetas `contenido` deben tener un atributo `id` de tipo `ID`. Lo mismo le debe suceder a los subelementos referenciados desde una etiqueta `contenidoGrupo`.
  - Los espacios en blanco de los elementos referenciado desde cualquier etiqueta de `contenidoPanel` son tenidos en cuenta.
  - En los contenidos, los atributos de tipo `ID` que sean destinos de enlaces deben llamarse `id`, y solamente puede utilizarse este nombre para este tipo de atributos. De esta forma GAP puede encontrarlos y utilizarlos como nombre de las páginas HTML generadas, con el fin de que sean referenciadas.
  - Las anclas origen deben tener como nombre de atributo origen `href` de tipo `IDREF`.
  - Las anclas destino deben tener como nombre de atributo destino `name` de tipo `CDATA`. Para referenciarlas debe utilizarse la nomenclatura `href="idEleto#ancla"`, donde `idEleto` es el valor del atributo `id` de tipo `ID` del elemento que contiene al elemento con el atributo `name`.
  - Las imágenes deben ser referenciadas a través de un atributo `imagen` de tipo `CDATA`.
  - Para incluir anclas en las imágenes se debe seguir un esquema:

```
<nombreEleto1 imagen="foto.jpg">
          <nombreEleto2 coords="c1" href="r1"/>
           ..............
          <nombreEleto2 coords="cn" href="cn"/>
</nombreEleto1>
```

    para obtener:

```
<img src="foto.jpg" usemap="#foto.jpg"/>
 <map>
   <area shape="poly" coords="c1" href="r1"/>
    .......
   <area shape="poly" coords="cn" href="rn"/>
 </map>
```

  - Las anclas origen que tengan como destino un proceso subordinado deben tener como nombre de atributo origen `hrefP` de tipo `CDATA`.

Todas estas restricciones en el documento de contenidos sirven para que funcione correctamente la transformación genérica XSLT que genera los contenidos a partir del documento de contenidos.

Transformación XSLT

Es la encargada de traducir los contenidos en formato XML (las partes de estos seleccionadas mediante la expresión XPath correspondiente) a páginas HTML. Está formada por seis reglas:

- La primera cambia el elemento raíz por `html` y fija las características de presentación (`/` → `html`).
- La segunda selecciona los elementos con atributo `href` y genera un ancla origen (`*/@href` → `a/@href`).
- La tercera selecciona los elementos con atributo `name` y genera un ancla destino (`*/@name` → `a/@name`). Como en los contenidos se utiliza la nomenclatura `href="destino#ancla"`, la transformación genera `href="destino#ancla.html"` que no es un URL. GAP se encarga de solucionarlo con la función `colocarAlmoadilla()`.
- La cuarta selecciona los elementos con atributo imagen y genera los elementos `img` y `map`, asignando como nombre de mapa el del nombre de la imagen (`*/@imagen` → `img/@src` + `img/@usemap` + `map`).
- La quinta selecciona los elementos con atributo `href` y con atributo `coords` para generar los elementos `area` (`*/@href` & `@coords` → `area`).
- La sexta selecciona los elementos con atributo hrefP y genera un ancla origen (`*/@hrefP` → `a/@href`).

La Figura B.13 muestra la transformación XSLT (archivo `c:/plumbingXJ/xml/trans.xsl`).

```
<?xml version="1.0" encoding="ISO-8859-1"?>


<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output method="html"
            encoding = "ISO-8859-1"
            omit-xml-declaration="yes"
            standalone="yes"
                    doctype-public=""
                    doctype-system=""
                    cdata-section-elements=""
                    indent="no"
                    media-type=""/>


<xsl:template match="/">
<html>
  <body>
  <xsl:apply-templates/>
  </body>
</html>
</xsl:template>

<xsl:template match="*[@href]">
  <a href="{@href}.html" style= "font-style:italic; color:black; text-
decoration:none">
    <xsl:apply-templates/>
  </a>
</xsl:template>

<xsl:template match="*[@name]">
  <a name="{@name}">
   <xsl:apply-templates/>
  </a>
</xsl:template>

<xsl:template match="*[@imagen]">
  <xsl:choose>
    <xsl:when test= "./*[@coords]">
    <img src= "file:{@imagen}" usemap= "#{@imagen}"></img>
      <map name="{@imagen}">
        <xsl:apply-templates/>
      </map>
    </xsl:when>
    <xsl:otherwise>
     <img src= "file:{@imagen}"></img>
      <xsl:apply-templates/>
     </xsl:otherwise>
  </xsl:choose>
 </xsl:template>

<xsl:template match="*[@href][@coords]">
  <area shape= "poly" coords= "{@coords}" href="{@href}.html" />
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*[@hrefP]">
 <a href="{@hrefP}" style= "font-style:italic; color:black; text-decoration:none">
   <xsl:apply-templates/>
 </a>
</xsl:template>

</xsl:stylesheet>
```

Figura B.13 Transformación XSLT

No se optó por utilizar enlaces XLink debido a:

- GAP solo considera enlaces simples.
- Como la transformación opera sobre partes de documentos, esto obligaba a incluir el espacio de nombres en cada subarchivo XML de entrada a la transformación.
- Se incluyen anclas en imágenes, no estando cubierto este supuesto por el estándar XLink.

Potencia expresiva de GAP

La potencia expresiva de GAP es inferior a la proporcionada por Pipe. En particular GAP no es capaz de:

- Representar contenidos dinámicos
- Representar enlaces n-arios.
- Utilizar dos tuberías distintas con origen en el mismo nodo.
- Ignorar enlaces de contenidos.
- Soportar las capacidades de sincronización de Pipe.
- Incluir todos los tipos de contenidos considerados por Pipe.

Veamos cuales pueden sor posibles soluciones, a tener en cuenta como trabajo futuro. Respecto a la primera limitación la línea de investigación debería ser similar a la propuesta para la inclusión de procesos subordinados (ver Apéndice B). La segunda es un problema ligado a las tecnologías subyacentes en Pipe. Una solución posible es la de generar un enlace cuyo destino fuese una página auxiliar que contuviese todos los enlaces los cuales deberían ser seleccionados por el usuario. La tercera limitación obliga a identificar que enlace está canalizado por cada tubería que parte del mismo nodo. Por lo tanto se necesitaría poder asignarlo enlace por enlace, y guardar dicha lista para gestionar la activación. La cuarta pasaría por una enumeración explícita de los enlaces de contenidos que se quieren obviar. La quinta obliga a una gestión del tiempo por parte de GAP para actualizar la aplicación. Finalmente, la sexta se resuelve incluyendo la etiqueta correspondiente y su transformación a su etiqueta funcional HTML.

Todas estas limitaciones se consideran como trabajo futuro de esta tesis.

Directorios

GAP funciona dentro del directorio `c:\plumbingXJ\GAP`. Hay cuatro directorios dentro de este.

- `clases` contiene las clases Java.
- `documentos` contiene los documentos de contenidos y de la aplicación, así como sus DTDs.
- `xml` contiene los archivos intermedios XML generados. Además también contiene a `transf.xsl`.
- `html` contiene los archivos HTML generados para el funcionamiento de la aplicación. Además también contiene a `blanca.html`.

Observaciones

Hay un par de observaciones a hacer respecto a GAP y que decidimos incluir en este apartado.

- Por alguna razón, JDK 1.3.0 no cambia el puntero del ratón a una "mano de selección" cuando pasa por encima del ancla origen de un hiperenlace establecida en una figura. De todas formas, si que reacciona bien a la selección de dicha ancla.
- GAP no lleva cuenta de si el tamaño de los paneles excede el tamaño máximo de la pantalla (1024x768, normalmente). En caso de exceder ese tamaño, la aplicación se genera, aunque en función del sistema operativo puede dar mayores o menores problemas. Por ejemplo, en Windows 98 "corta los paneles". En Windows XP si se genera la aplicación, pero las imágenes no se cargan en los paneles al menos que estos se refresquen (por ejemplo recolocándolos).
- Asimismo, si un contenido incluye una imagen que no cabe en el panel que la muestra, será necesario refrescar el panel (mover el scroll) para que se muestre la imagen.

Como podemos ver no son cuestiones claves al diseño de GAP, pero hemos decidido incluirlas a título informativo.

B.3 Código

Aplicacion.java

```
import javax.swing.*;
import javax.swing.event.*;

import java.awt.*;
import java.awt.event.*;

import java.lang.ArrayIndexOutOfBoundsException;
import org.apache.xerces.parsers.DOMParser;
import org.xml.sax.SAXException;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Element;
import org.xml.sax.helpers.LocatorImpl;

import Error;


public class Aplicacion {

    private Ventana ventana []; //el esquema navegacional de la aplicación
    private int numVent; //el número de ventanas de la aplicación
    private Document docAplic; //el documento de la aplicación
    private Document docCont; //el documento de contenidos

    private final static String directorio= "c:\\plumbingXJ\\GAP\\documentos\\"; //el directorio actual

    //Aplicacion recibe el nombre de los archivos del documento de la aplicación y el de contenidos
    public Aplicacion(String archDocApS, String archDocContS) throws Exception
    {
        //lectura documento de la aplicacion
        DOMParser parser= new DOMParser();
        parser.setErrorHandler(new DPErrorHandler());

        parser.setFeature("http://xml.org/sax/features/validation", true);
        parser.setFeature("http://apache.org/xml/features/dom/include-ignorable-whitespace", true);
```

```
        parser.parse("file:" + directorio + archDocApS);
    docAplic= parser.getDocument();

    parser.parse("file:" + directorio + archDocContS);
    docCont= parser.getDocument();



    //las ventanas de la aplicación
    NodeList ventanasXML= docAplic.getElementsByTagName("ventana");
    numVent= ventanasXML.getLength();
    ventana= new Ventana [numVent + 1];


    //inicializamos las ventanas, y por tanto la aplicación
    for (int i=1; i<=numVent; i++) //obviamos la cero, para que se correspondan los id de las ventanas con su posición en el array
    { ventana[i]= new Ventana(this, (Element) ventanasXML.item(i-1), docCont);
      ventana[i].addWindowListener(new WindowAdapter()
                                   { public void windowClosing(WindowEvent e)
                                       { System.exit(0); }
                                   });
      ventana[i].setLocation(200, 200);
    }
    //la pantalla inicial es la pantalla 1 (obviamos la 0)

           ventana[1].activar();
    ventana[1].pack();
    ventana[1].setVisible(true);
}


   //activa una ventana al inicializar o al ser seleccionada por un activadorV
   public void activarVentana(int nVen)
   {  try { ventana[nVen].activar();
         } catch (ArrayIndexOutOfBoundsException e) { Error.mensaje("Ventana o panel mal numerado");
         } catch (NullPointerException e) { Error.mensaje("Ventana o panel mal numerado"); //la posición 0 no tiene objeto
         } catch (Exception e) //para no estropear el controlador de eventos de ActivadorV
           { Error.mensaje (e.getMessage()); }
   }



   //activa una ventana como resultado de la activacion de un enlace
```

```
public void activarVentana(int nVen, int nPan, String contenido)
{ try { ventana[nVen].activar(nPan, contenido);
      } catch (ArrayIndexOutOfBoundsException e) { Error.mensaje("Ventana o panel mal numerado");
      } catch (NullPointerException e) { Error.mensaje("Ventana o panel mal numerado"); //la posición 0 no tiene objeto
      } catch (Exception e) //para no estropear el controlador de eventos de ActivadorV
        { Error.mensaje (e.getMessage()); }
 }


//muetra el contenido seleccionado por un enlace en un panel
public void mostrarContenido(int nVen, int nPan, String contenido)
{ try { ventana[nVen].mostrarContenido(nPan, contenido);
      } catch (NullPointerException e) { Error.mensaje("Ventana o panel mal numerado");//la posición 0 no tiene objeto
      } catch (ArrayIndexOutOfBoundsException e) { Error.mensaje("Ventana o panel mal numerado");
      } catch (Exception e) //para no estropear el controlador de eventos de ActivadorV
        { Error.mensaje (e.getMessage()); }
 }


//para los procesos de una ventana
public void salirVentana(int nVen)
{ try { ventana[nVen].salir();
      } catch (NullPointerException e) { Error.mensaje("Ventana o panel mal numerado"); //la posición 0 no tiene objeto
      } catch (ArrayIndexOutOfBoundsException e) { Error.mensaje("Ventana o panel mal numerado");}
}


//oculta una ventana
public void ocultarVentana(int nVen)
{ try { ventana[nVen].setVisible(false);
      } catch (NullPointerException e) { Error.mensaje("Ventana o panel mal numerado"); //la posición 0 no tiene objeto
      } catch (ArrayIndexOutOfBoundsException e) { Error.mensaje("Ventana o panel mal numerado");  throw e;}
}



//actualiza los componentes de una ventana
void empaquetarVentana(int nVen)
{ try { ventana[nVen].pack();
      } catch (NullPointerException e) { Error.mensaje("Ventana o panel mal numerado"); //la posición 0 no tiene objeto
      } catch (ArrayIndexOutOfBoundsException e) { Error.mensaje("Ventana o panel mal numerado");  throw e;}
}
```

```
    public static void main(String args[])
    {
              //obtenemos el nombre de los archivos, y se los pasamos a Aplicación
        final JFrame pAux= new JFrame("PlumbingXJ");
        JPanel pCont= new JPanel();
        JPanel datos= new JPanel();
        JPanel botonera= new JPanel();
        pCont.setLayout(new BoxLayout(pCont, BoxLayout.Y_AXIS));
        datos.setLayout(new BoxLayout(datos, BoxLayout.Y_AXIS));
        JLabel etiqueta1= new JLabel("Documento de la aplicación:");
        JLabel etiqueta2= new JLabel("Documento de contenidos:");
            final JTextField campo1= new JTextField(20);
               final JTextField campo2= new JTextField(20);
        JButton aceptar= new JButton("Aceptar");
        JButton salir= new JButton("Salir");
        aceptar.setPreferredSize(new Dimension (80, 25));
        salir.setPreferredSize(new Dimension (80, 25));
        datos.add(etiqueta1);
        datos.add(campo1);
        datos.add(etiqueta2);
        datos.add(campo2);
        botonera.add(aceptar);
        botonera.add(salir);
        pCont.add(datos);
        pCont.add(botonera);
        aceptar.addActionListener(new ActionListener()
                                                                    { public void actionPerformed(ActionEvent e)
                                                                           {   pAux.setVisible(false);
                                                                               //todas las excepciones que se lanzan en la E/S y no han sido
capturadas se mandan aquí
                                                                                  try {
                                                                                       Aplicacion     a=     new     Aplicacion(campo1.getText(),
campo2.getText());
                                                                                 } catch (RuntimeException ee) { Error.mensaje(ee.getMessage());
                                                                                 } catch (Exception ee) {Error.mensaje(ee.getMessage()); }
                                                                           }
                                                                     });
        salir.addActionListener(new ActionListener()
                                                                    { public void actionPerformed(ActionEvent e)
                                                                        { System.exit(0); }
                                                                     });
```

```
        pAux.setContentPane(pCont);
         pAux.addWindowListener(new WindowAdapter()
                                    { public void windowClosing(WindowEvent e)
                                        { System.exit(0); }
                                    });
                pAux.setLocation(200, 200);
                pAux.pack();
                pAux.setVisible(true);
        }
}
```

Ventana.java

```
import javax.swing.*;
import javax.swing.event.*;

import java.awt.*;
import java.awt.event.*;

import java.net.URL;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Attr;
import org.w3c.dom.NodeList;


//una ventana es un JFrame
class Ventana extends JFrame {

    private Panel panel[]; //los paneles contenidos en esta ventana
    private ActivadorV activadorV[]; //los activadores de ventanas contenidos en esta pantalla
    private int numPaneles, numActivadoresV, idVent;// el número de paneles, activadores de ventanas y el identificador de la ventana

    //Ventana recibe el esquema navegacional, la ventana en formato XML, y el documento de contenidos
    public Ventana (Aplicacion aplicacion, Element ventanaXML, Document docCont) throws Exception
    {

        //ponemos titulo a la ventana
         NodeList titulosXML= ventanaXML.getElementsByTagName("nombre");
         if (titulosXML.getLength()>0)
```

```
    { Element tituloXML= (Element) titulosXML.item(0);
      setTitle(tituloXML.getFirstChild().getNodeValue());
    }

          //creamos los elementos navegacionales de la ventana
       Attr idVentA= ventanaXML.getAttributeNode("id");
    int idVent= Integer.valueOf(idVentA.getValue().substring(1)).intValue();
    NodeList panelesXML= ventanaXML.getElementsByTagName("panel");
    NodeList activadoresVXML= ventanaXML.getElementsByTagName("activadorV");
    numPaneles= panelesXML.getLength();
    numActivadoresV= activadoresVXML.getLength();
    JPanel botonera= new JPanel();
    if (numActivadoresV >0)
       botonera.setLayout(new BoxLayout(botonera, BoxLayout.Y_AXIS));
    getContentPane().setLayout(new BoxLayout(getContentPane(), BoxLayout.X_AXIS));
    panel= new Panel [numPaneles+1]; //obviamos el 0 para hacer más legible el código
    activadorV= new ActivadorV [numActivadoresV+1]; //obviamos el 0 para hacer más legible el código
    for (int i= 1;i <= numPaneles; i++)
     {
       panel[i]= new Panel(aplicacion, idVent, (Element) panelesXML.item(i-1), docCont);
       getContentPane().add(panel[i]);
     }
    for (int i= 1; i <= numActivadoresV; i++)
    {
      activadorV[i]= new ActivadorV(aplicacion, idVent, (Element) activadoresVXML.item(i-1));
      botonera.add(activadorV[i]);
    }
    if (numActivadoresV>0) getContentPane().add(botonera);//si hay activadores de ventanas añadimos la botonera a la ventana
 }

public void activar() throws Exception //enlace proveniente de un boton o inicializacion
 {
    for (int i= 1; i<= numPaneles; i++)
     panel[i].mostrarDefecto();
    pack();
    setLocation(0, 0);
    setVisible(true);
 }

 public void activar(int idPanel, String contenido) throws Exception  //enlace proveniente de un nodo contenedor de otro nexo
 {
    for (int i= 1; i<= numPaneles; i++)
     if (i != idPanel) panel[i].mostrarDefecto(); //si no se hace asi se pueden activar procesos costosos
```

```
                panel[idPanel].cargarContenido(contenido);
            pack();
            setLocation(0, 0);
            setVisible(true);
        }


        public void mostrarContenido(int idPanel, String contenido) throws Exception //para enlaces que provienene de un nodo contenedor del mismo
nexo
        {  panel[idPanel].cargarContenido(contenido); }

        public void salir()
        {
            for (int i=1; i<=numPaneles; i++)
              panel[i].salir();
            this.setVisible(false);
        }

}
```

Panel.java

```
import javax.swing.*;
import javax.swing.event.*;

import java.awt.*;
import java.awt.event.*;

import java.net.URL;
import java.lang.String;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.FileInputStream;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.Attr;
import org.w3c.dom.NodeList;
import org.w3c.dom.traversal.NodeIterator;
```

```
import org.apache.xpath.XPathAPI;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;

import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerConfigurationException;

import ProcesoSubordinado;
import java.lang.Object;
import java.lang.Class;
import java.lang.reflect.Constructor;
import java.lang.ClassNotFoundException;
import java.lang.NoSuchMethodException;
import java.lang.InstantiationException;
import java.lang.IllegalAccessException;
import java.lang.reflect.InvocationTargetException;

import Error;


//Un Panel es un JPanel
public class Panel extends JPanel {

    private Aplicacion aplicacion; // acceso a las ventanas que componen la aplicacion
    private int ventanaContenedora; // la Ventana que contiene al Panel
    private int ventanaDestino; // la Ventana destino del enlace que parte de este Panel (si hay)
    private int panelDestino;// el Panel destino del enlace que parte de este Panel (si hay)
    private String contDefecto; // contenido por defecto. Dirección de la hoja HTML, o nombre de la clase del proceso subordinado contenido por
defecto
    private JEditorPane panelHTML; // el panel donde se muestran los contenidos no procesos
       private JScrollPane panelScHTML; //el anterior con scroll
       private JPanel panelProceso; //el panel para el proceso. No cabe especializar Panel, ya que puede contener páginas o procesos
       private ProcesoSubordinado proceso; //proceso subordinado que se esta mostrando, si lo hay.
       private boolean eraProceso;

       private final static String directorioXML = "c:\\plumbingXJ\\GAP\\XML\\"; //el directorio de los archivos XML intermedios
       private final static String directorioHTML = "c:\\plumbingXJ\\GAP\\HTML\\"; //el directorio de los archivos HMTL generados
       private final static String packagePS = "plumbingXJ.GAP.documentos."; //el package donde se encuentran las clases de los procesos
subordinados
```

```
    // Un Panel recibe la aplicacion, el id. de la Ventana que la contiene, el Panel en formato XML y el documento de contenidos
    public Panel (Aplicacion a, int idVent, Element panelXML, Document docCont) throws Exception
    {
        aplicacion= a;
        ventanaContenedora= idVent;

        Node nodoXPath= null;

        String tam= panelXML.getAttribute("tam");
        int x= obtenerX(tam);
        int y= obtenerY(tam);

        //generación del contenido por defecto de un panel
        NodeList listaNodos= panelXML.getElementsByTagName("contenidoDefecto");
               if (listaNodos.getLength() > 0) //hay contenido por defecto
                       {       Element defectoXML= (Element) listaNodos.item(0);
                               String defecto= defectoXML.getFirstChild().getNodeValue();
                               String tipoDefecto= defectoXML.getAttribute("tipo");
                               if (tipoDefecto.equals("XPath"))
                               {   nodoXPath= XPathAPI.selectSingleNode(docCont, defecto); //el contenido está univocamente determinado al ser cte
                                       if (nodoXPath == null) Error.mensaje("La expresión XPath " + defecto + " en una etiqueta contenidoDefecto no
localiza ningún contenido");
                       else {  eraProceso= false;
                                       proceso= null;
                                              panelHTML= new JEditorPane();
                                               panelHTML.setEditable(false);
                                               panelHTML.addHyperlinkListener(new CEPanel());
                                               panelScHTML= new JScrollPane(panelHTML);
                                               panelScHTML.setPreferredSize(new Dimension (x, y));
                                       add(panelScHTML);

                                       String  nombreArchivoXML=  panelXML.getAttribute("id").replace('.',  '_')  +  ".xml";  //la  entrada  de  la
transformación
                                       String  nombreArchivoHTML=  panelXML.getAttribute("id").replace('.',  '_')  +  ".html";  //la  salida  de  la
transformación
                                                       generarHTML(nodoXPath, nombreArchivoXML, nombreArchivoHTML);

                                       contDefecto= "file:" + directorioHTML + nombreArchivoHTML;
                                }
                }
                else if (tipoDefecto.equals("proceso")) { eraProceso= true;
                                                                                        contDefecto= defecto;
```

```
                                                                                                                    }
            }
        else {  eraProceso= false;
                    proceso= null;
                panelHTML= new JEditorPane();
                            panelHTML.setEditable(false);
                            panelHTML.addHyperlinkListener(new CEPanel());
                            panelScHTML= new JScrollPane(panelHTML);
                            panelScHTML.setPreferredSize(new Dimension (x, y));
                add(panelScHTML);

                contDefecto= "file:"  + directorioHTML + "blanca.html"; //contenido null
            }

               //generacion de contenidos HTML destino de un enlace de contenidos en un panel (formato contenido)
        listaNodos= panelXML.getElementsByTagName("contenido");
        int numContenido= listaNodos.getLength();
        for (int i=0; i<numContenido; i++) //procesamos todos los contenidos asociados a un nodo
                        {       Element contenidoXML= (Element) listaNodos.item(i);
                                String tipoContenido= contenidoXML.getAttribute("tipo");
                         String contenido= contenidoXML.getFirstChild().getNodeValue();
                                if (tipoContenido.equals("XPath"))
                                {   nodoXPath= XPathAPI.selectSingleNode(docCont, contenido); //el contenido está univocamente determinado al ser cte
                                     if (nodoXPath==null) Error.mensaje("La expresión XPath " + contenido + " una etiqueta contenido no localiza ningún
contenido");
                        else {
                                                String idEleto= ((Element) nodoXPath).getAttribute("id");
                                                if (idEleto == "") Error.mensaje("El contenido localizado por la expresión XPath " + contenido + " en
una etiqueta contenido no tiene atributo id");
                                                else {  String nombreArchivoXML= idEleto + ".xml"; //la entrada de la transformación
                                                                String nombreArchivoHTML= idEleto + ".html"; //la salida de la transformación
                                        generarHTML(nodoXPath, nombreArchivoXML, nombreArchivoHTML);
                                    }
                                            }
                    }
                         }


               //generación de los contenidos HTML destino de un enlace de contenidos de un panel (formato grupo de contenidos)
        listaNodos= panelXML.getElementsByTagName("contenidoGrupo");
        int numContenidoG;
        numContenidoG= listaNodos.getLength();
        for (int i=0; i<numContenidoG; i++)
```

```
        {      Element contenidoXML= (Element) listaNodos.item(i);
               String XPath= contenidoXML.getFirstChild().getNodeValue() + "/*"; //para obtener los hijos del elemento que los agrupa
               NodeIterator nodosXPath= null;
               nodosXPath= XPathAPI.selectNodeIterator(docCont, XPath); // lista de nodos correspondientes a los contenidos del grupo
               if (nodosXPath.nextNode() == null) Error.mensaje ("La expresión XPath " + XPath +" en una etiqueta contenidoGrupo no localiza nigún
contenido");
                     else {   nodosXPath= XPathAPI.selectNodeIterator(docCont, XPath); //deja el iterador al principio
                              while ((nodoXPath= nodosXPath.nextNode()) != null)
                                        { String idEleto= ((Element) nodoXPath).getAttribute("id");
                                             if (idEleto == "") Error.mensaje("El subelemento localizado por la expresión XPath " + XPath + " en
una etiqueta contenidoGrupo no tiene atributo id");
                                                       else {  String nombreArchivoXML= idEleto + ".xml"; // funciona ya que es un identificador
                                                       String nombreArchivoHTML= idEleto + ".html"; // funciona ya que es un identificador
                                                       generarHTML(nodoXPath, nombreArchivoXML, nombreArchivoHTML);
                                          }
                                        }
                     }
         }

               // generación de la tubería
        listaNodos= panelXML.getElementsByTagName("destinoEnlaces"); // el destino del hiperenlace (si hay)
        if (listaNodos.getLength() > 0) //realmente hay tuberia
        { Element destinoXML= (Element) listaNodos.item(0);
          Attr destinoA= destinoXML.getAttributeNode("panel");
          String destinoS= destinoA.getValue();

          if (obtenerNumVentana(destinoS) == idVent) //es un panel de la misma ventana
               { ventanaDestino= idVent;
                 panelDestino= obtenerNumPanel(destinoS);
               }
          else { ventanaDestino= obtenerNumVentana(destinoS); //es un panel de otra ventana
                        panelDestino= obtenerNumPanel(destinoS);
                  }

        }
    }

       //genera la página HTML a partir de un nodo XPath
       private void generarHTML(Node nodoXPath, String nombreArchivoXML, String nombreArchivoHTML) throws Exception
       {
               String rutaArchivoXML= directorioXML + nombreArchivoXML;
               String rutaArchivoHTML= directorioHTML + nombreArchivoHTML;
```

```
            // fijamos el formato de salida para el XMLSerializer
         OutputFormat formatoSalida= new OutputFormat();
         formatoSalida.setEncoding("ISO-8859-1");
         formatoSalida.setPreserveSpace(true);

        FileOutputStream archivo= new FileOutputStream(rutaArchivoXML);
                XMLSerializer serializador= new XMLSerializer(archivo, formatoSalida);
        serializador.serialize((Element) nodoXPath);// generamos la entrada a la transformación
        archivo.close();

         //aplicacion de la transformacion XSLT
                try { TransformerFactory tFactory= TransformerFactory.newInstance();
                  Transformer transformador= tFactory.newTransformer(new StreamSource(new FileInputStream(directorioXML + "transf.xsl")));
                            transformador.transform(new      StreamSource(new      FileInputStream(rutaArchivoXML)),      new      StreamResult(new
FileOutputStream(rutaArchivoHTML)));
                } catch(TransformerException e) { Error.mensaje("Error en la transformación XSLT " + directorioXML + "trans.xsl"); } // por si se
borra transf.xsl

        }


        private int obtenerX(String cad)
        {
         int lon, i= 0;
                StringBuffer aux1= new StringBuffer();

                lon= cad.length();
                while (i<lon && cad.charAt(i) != ',')
                 if (cad.charAt(i) != ' ') aux1.append(cad.charAt(i++));
                 else i++;
                return Integer.valueOf(new String(aux1)).intValue();
        }


        private int obtenerY(String cad)
        {
                int lon, i= 1, j;
                StringBuffer aux1= new StringBuffer();

                lon= cad.length();
                while (i<lon && cad.charAt(i) != ',')
                 i++;
```

```
            i++; //saltar coma

            for (j= i; j<lon; j++)
             if (cad.charAt(j) != ' ') aux1.append(cad.charAt(j));

            return Integer.valueOf(new String(aux1)).intValue();

    }

    private int obtenerNumVentana(String cad)
    {
            int lon, i= 1;
            StringBuffer aux1= new StringBuffer();

            lon= cad.length();
            while (i<lon && cad.charAt(i) != '.')
             aux1.append(cad.charAt(i++));

            return Integer.valueOf(new String(aux1)).intValue();

    }

    private int obtenerNumPanel(String cad)
    {
            int lon, i= 1, j;
            StringBuffer aux1= new StringBuffer();

            lon= cad.length();
            while (i<lon && cad.charAt(i) != '.')
             i++;

            i++; //saltar punto

            for (j= i; j<lon; j++)
             aux1.append(cad.charAt(j));

            return Integer.valueOf(new String(aux1)).intValue();

    }

//para la activación de una ventana
public void mostrarDefecto() throws Exception
{
```

```
   if (esArchivo(contDefecto)) cargarPagina(contDefecto);
   else activarProceso(contDefecto);

}


//carga un contenido a petición de activar() o tras seleccionarse un enlace
   public void cargarContenido(String contenido) throws Exception
   {  if (esArchivo(contenido)) cargarPagina(contenido);
      else activarProceso(quitarURL(contenido));

   }


   //carga una página HTML en el JEditorPane
private void cargarPagina(String nombrePagina) throws Exception
{
  if (eraProceso) //enlace proveniente de otra ventana
     { add(panelScHTML);
       aplicacion.empaquetarVentana(ventanaContenedora); }

  if (proceso != null) //enlace proveniente de la misma ventana
     { proceso.parar();
             proceso= null;
       remove(panelProceso);
       add(panelScHTML);
       aplicacion.empaquetarVentana(ventanaContenedora); }

  URL URLAux= new URL(nombrePagina);
     try { panelHTML.setPage(URLAux);
         } catch (Exception e) { Error.mensaje("No se encuentra la página " + e.getMessage()); } //por si se borra blanca.html

}


   //activa un proceso
   private void activarProceso(String nombreC)
          {
             if (!eraProceso)  remove(panelScHTML); //enlace proveniente de otra ventana

             if (proceso != null) //enlace proveniente de la misma ventana
                    { proceso.parar();
                   proceso= null;
```

```
                        remove(panelProceso);}

          panelProceso= new JPanel();

          //carga de la clase especificada en nombreC
          try {    String nombreClase= nombreClase=  quitarExtension(nombreC); //si la activación proviene de un hiperenlace hay que quitarle el
.ps
                                Class clase= Class.forName(packagePS + nombreClase);
                                   Constructor constructor= clase.getConstructor(new Class [] {});
                                   Object objeto= constructor.newInstance(new Object [] {});
                                   proceso= (ProcesoSubordinado) objeto;
                              } catch (ClassNotFoundException e) { Error.mensaje("Error en un atributo hrefP. No se encuentra la clase: " +
nombreC);
                              } catch (NoSuchMethodException e)   { Error.mensaje("Error en la clase " + nombreC + ". No existe un constructor
adecuado");
                              } catch (InstantiationException e) { Error.mensaje("La clase" + nombreC + " es abstracta");
                              } catch (IllegalAccessException e) { Error.mensaje("Error en la clase " + nombreC + ". Constructor privado");
                              } catch (InvocationTargetException e) { Error.mensaje("Error en la clase" + nombreC + ". El constructor lanzó una
excepción");  }
                  proceso.fijarPanel(panelProceso);
                  proceso.activar();
                  add(panelProceso);
                  aplicacion.empaquetarVentana(ventanaContenedora);

              }


    //la función de activación de enlaces
    public class CEPanel implements HyperlinkListener {
        public void hyperlinkUpdate(HyperlinkEvent e)
        { if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
          {  salir();
             String contenido= new String(colocarAlmoadilla(e.getURL().toString())); //por si lleva un ancla "mal colocada" por la transformación
XSLT

             if (ventanaDestino == ventanaContenedora) // el destino es un panel de la misma ventana
                  aplicacion.mostrarContenido(ventanaContenedora, panelDestino, contenido); // mostramos el contenido del nodo contenedor que
recibe el enlace
             else //el destino es un panel de otra ventana
                 { aplicacion.salirVentana(ventanaContenedora); // ocultamos la ventana actual y paramos sus procesos
                   aplicacion.activarVentana(ventanaDestino, panelDestino, contenido); // se activa la ventana con el contenido seleccionado por
el enlace en el panel correspondiente
                 }
```

```
      }
    }
 }


public void salir()
    { if (proceso != null) { proceso.parar();
                                                    proceso= null; //el garbage colector invoca a finalize()
                                                remove(panelProceso);
                                                eraProceso= true;
                                    }
      else eraProceso= false; //no quitamos el JEditorPane porque se puede reutilizar en el mejor de los casos para cargar más páginasz

    }


private String colocarAlmoadilla (String cad) //convierte aa#bb.cc en aa.cc#bb La mayoría de las veces deja aa.bb en aa.bb
{ int lon, i;
  StringBuffer aux1= new StringBuffer(200);
  StringBuffer aux2= new StringBuffer(100);
  boolean sacoAncla= false;
  lon= cad.length();

  for (i= 0; i<lon; i++)
    if (cad.charAt(i) == '#') { sacoAncla= true; aux2.append(cad.charAt(i)); }
    else if (cad.charAt(i) == '.') { sacoAncla= false; aux1.append(cad.charAt(i)); }
         else if (sacoAncla) aux2.append(cad.charAt(i));
              else  aux1.append(cad.charAt(i));

  aux1.append(aux2);

  return aux1.toString();

}


private String quitarExtension (String cad) //convierte aa.bb en aa
           {
                  int lon= cad.length();
                  if (lon != 0)
                        { int i= lon-1;
                                 while (i > 0 && cad.charAt(i) != '.') i--;
                                 if (i != 0) return cad.substring(0, i);
```

```
                                    else return cad;
                                    }
                        else return cad;
                }


        private boolean esArchivo (String cad) //detecta cadenas aa.bb
        {       int lon= cad.length();
                if (lon != 0)
                     {  int i=0;
                             while (i<lon && cad.charAt(i) != '.') i++;
                             return (i < lon);
                      }
                else return false;
        }


    private String quitarURL(String cad) //convierte file://aa.bb en aa.bb
        { int lon=cad.length();
          int i= lon - 1;

          while (i>=0 && cad.charAt(i) != '/') i--;

          return cad.substring(++i, lon);
         }




}
```

ActivadorV.java

```
import javax.swing.*;
import javax.swing.event.*;

import java.awt.*;
import java.awt.event.*;

import java.net.URL;
```

```
import org.w3c.dom.NodeList;
import org.w3c.dom.Element;
import org.w3c.dom.Attr;


// Un activador de ventana es un JButton
public class ActivadorV extends JButton {

    private Aplicacion aplicacion; //acceso a las ventanas de la aplicación
    private int ventanaContenedora; // la ventana que contiene al activador de ventana
    private int ventanaDestino; // la ventana destino del enlace del activador de ventana


    // un ActivadorV recibe la aplicacion, la ventana que lo contiene, y el activador de ventana en formato XML
    public ActivadorV (Aplicacion a,  int idVent, Element activadorVXML)
    {
        aplicacion= a;
        ventanaContenedora= idVent;
        addActionListener(new CEActivadorV());

        NodeList listaNodos= activadorVXML.getElementsByTagName("nombre");
       Element nombreXML= (Element) listaNodos.item(0);
       String nombre= nombreXML.getFirstChild().getNodeValue();
        setText(nombre); // el nombre del activador de ventana

        Attr destinoXML= activadorVXML.getAttributeNode("ventana");
        ventanaDestino= Integer.valueOf(destinoXML.getValue().substring(1)).intValue();

     }

    public class CEActivadorV implements ActionListener {
            public void actionPerformed(ActionEvent e)
                                    {       aplicacion.salirVentana(ventanaContenedora);
                                            aplicacion.ocultarVentana(ventanaContenedora); // se oculta la ventana actual
                                            aplicacion.activarVentana(ventanaDestino);  // se hace visible la ventana destino
                                                                                                    //  nótese  que  esta  debe  tener  algún
contenido por defecto para algún panel
                                    }
       }
}
```

ProcesoSubordinado.java

```
import javax.swing.JPanel;


public interface ProcesoSubordinado {

    public void fijarPanel(JPanel p); //fija el panel que el proceso subordinado utilizará para construir su interfaz de usuario
    public void activar (); //muestra en panel los elementos de su interfaz
       public void parar(); //detiene cualquier actividad computacional que deba interrumpirse convenienemente antes de destruir el objeto
       }
```

DPErrorHandler.java

```
import org.xml.sax.HandlerBase;
import org.xml.sax.SAXParseException;

public class DPErrorHandler extends HandlerBase {

public void error(SAXParseException exception) throws SAXParseException
{ throw exception; }

public void fatalError(SAXParseException exception) throws SAXParseException
{ throw exception; }

public void warning(SAXParseException exception) throws SAXParseException
{ throw exception; }


}
```

Error.java

```
import javax.swing.*;
import javax.swing.event.*;

import java.awt.*;
import java.awt.event.*;

import java.lang.String;
```

```
//tratamiento de excepciones
public class Error {

  public static void mensaje(String e)
        {
               JFrame pErr= new JFrame("Error");
         JEditorPane panelE= new JEditorPane();
         panelE.setEditable(false);
        JScrollPane panelSc= new JScrollPane(panelE);
         panelSc.setPreferredSize(new Dimension (600, 100));
         panelE.setText (e);
        pErr.getContentPane().add(panelSc);
        pErr.addWindowListener(new WindowAdapter()
                                        { public void windowClosing(WindowEvent e)
                                                                        { System.exit(0); }
                                        });
        pErr.setLocation(200, 200);
        pErr.pack();
        pErr.setVisible(true);
   }
}
```

Apéndice C. Documentos GAP

C.1 Web del profesor

C.1.1 DTD de contenidos

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ENTITY % textoA "(#PCDATA|refA)*">
<!ELEMENT refA (#PCDATA)>
<!ATTLIST refA href CDATA #REQUIRED>

<!ELEMENT profesor (índice, biografía, españa, estudios, docencia, formulario,
localidades, asignaturas, resultado)>

<!ELEMENT índice (ancla+)>
<!ATTLIST índice imagen CDATA #REQUIRED>
<!ELEMENT ancla EMPTY>
<!ATTLIST ancla href CDATA #REQUIRED
                coords CDATA #REQUIRED>

<!ELEMENT biografía %textoA;>
<!ATTLIST biografía id ID #REQUIRED>

<!ELEMENT españa (ancla+)>
<!ATTLIST españa imagen CDATA #REQUIRED
                 id ID #REQUIRED>

<!ELEMENT estudios (#PCDATA)>
<!ATTLIST estudios id ID #REQUIRED>

<!ELEMENT docencia (#PCDATA|refA|desD)*>
<!ATTLIST docencia id ID #REQUIRED>
<!ELEMENT desD (#PCDATA)>
<!ATTLIST desD name CDATA #REQUIRED>

<!ELEMENT formulario %textoA;>
<!ATTLIST formulario id ID #REQUIRED>


<!ELEMENT localidades (localidad+)>
<!ELEMENT localidad (#PCDATA)>
<!ATTLIST localidad nombre CDATA #REQUIRED
                    id ID #REQUIRED>

<!ELEMENT asignaturas (asignatura+)>
<!ELEMENT asignatura (#PCDATA)>
<!ATTLIST asignatura nombre CDATA #REQUIRED
                     id ID #REQUIRED>

<!ELEMENT resultado %textoA;>
<!ATTLIST resultado id ID #REQUIRED>
```

C.1.2 Documento de contenidos

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE profesor SYSTEM "contenidosProfesorIndex.dtd">


<profesor>
  <índice imagen="c:\plumbingXJ\GAP\documentos\indice.jpg">
    <ancla href="espana" coords= "0 ,0, 108, 0, 108, 42, 0, 42, 0, 0"/>
    <ancla href="estudios" coords= "0, 42, 108, 42, 108, 84, 0, 84, 0, 42"/>
    <ancla href="docencia" coords= "0, 84, 108, 42, 108, 126, 0, 126, 0, 84"/>
    <ancla href="formulario" coords= "0, 126, 108, 126, 108, 168, 0, 168, 0, 127"/>
  </índice>

  <biografía id="biografia">Esta es una breve introducción a la vida de Antonio
```

```
Navarro, nacido, criado, y formado en <refA href="espana">España</refA>
Realice mis <refA href="estudios">estudios superiores</refA> en la Universidad
Complutense de Madrid.

En dicha universidad trabajo actualmente de profesor,
siendo mi <refA href="docencia">docencia actual:</refA> <refA
href="docencia#isg1">Ingeniería
del Software de Gestión I</refA>, <refA href="docencia#isg2">Ingeniería del Sofware de
Gestión II</refA>,
e <refA href="docencia#isg91">Ingeniería del Software 91</refA>.
Si deseas iniciar una búsqueda en este sitio Web pulsa <refA
href="formulario">aquí</refA>.
  </biografía>

  <españa imagen="c:\plumbingXJ\GAP\documentos\espana.gif" id="espana">
    <ancla href="madrid" coords="120, 120, 150, 100, 160, 130, 120, 120"/>
    <ancla href="lepe" coords="45, 230, 55, 235, 50, 240, 45, 230"/>
  </españa>

  <estudios id="estudios">Soy licenciado en Ciencias Matemáticas, especialidad de
Ciencias de la Computación. Lo que más me gusto de esta carrera es la combinación
de formación matemática e informática recibida, y que sin duda alguna influyo en mi
vida.
  </estudios>

  <docencia id="docencia">En la actualidad imparto una docencia exclusivamente centrada
en la asignatura de Ingeniería del Software, asignatura que también ha influido en mi
vida.

 A mi entender, esta es una de las asignaturas más bonitas de toda la carrera. En ella
se
presentan a los alumnos las técnicas actuales de gestión de proyectos y de análisis y
diseño
orientado a objetos. <desD name="isg1">En</desD> la asignatura de <refA href="isg1">
Ingeniería del Software de Gestión I</refA>, se pretende que el alumno aprenda a
desarrollar
planes de proyecto de software, indispensables para una correcta gestión
del proyecto. En particular se presta especial atención a los modelos de proceso, las
técnicas
de estimación, y la planificación temporal. <desD name="isg2">En</desD> la asignatura de
<refA href="isg2">Ingeniería del Software de Gestión II</refA>, se pretende que el
alumno sea
capaz de crear diseños orientados a objetos siguiendo las directrices propuestas por
diversas
metodologías. En particular se presenta la notación de diseño UML y una serie de
patrones
orientados a objetos.
A pesar de esto, las ideas de mayor nivel son las proporcionadas por Grady Booch en su
libro
Object-oriented analysys and design, y por la experiencia. <desD
name="isg91">Finalmente</desD>
en la asignatura de <refA href="isg91">Ingeniería del Software de Gestión 91</refA> se
da soporte
a los alumnos del extinto plan antiguo.
 </docencia>

  <formulario id="formulario">Se supone que aquí aparece un formulario de
búsqueda, con un campo de entrada y un <refA href="resultado">botón</refA>.
  </formulario>

  <localidades>
    <localidad nombre="Madrid" id="madrid">Madrid, Madrid, Madrid,
pedazo de la España en que nací</localidad>
    <localidad nombre="Lepe" id="lepe">Lepe, localidad de España
famosa por sus chistes.</localidad>
  </localidades>

  <asignaturas>
    <asignatura nombre="isg1" id="isg1">Ingeniería del Software de Gestión I,
es parte de la docencia que imparto.</asignatura>
    <asignatura nombre="isg2" id="isg2">Ingeniería del Software de Gestión II,
es parte de la docencia que imparto.</asignatura>
<asignatura nombre="isg91" id="isg91">Ingeniería del Software de Gestión 91,
es parte de la docencia que imparto.</asignatura>
```

```
  </asignaturas>

  <resultado id="resultado">Se supone que este es el resultado de la búsqueda
que enlaza con los contenidos. Por ejemplo con
<refA href="isg1">Ingeniería del Software de Gestión I</refA>,
<refA href="isg2">Ingeniería del Sofware de Gestión II</refA>,
y con <refA href="isg91">Ingeniería del Software de Gestión 91</refA>
  </resultado>

</profesor>
```

C.1.3 Documento de la aplicación

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">
<aplicacion id="ap1">

   <ventana id="v1">
      <nombre>Ventana 0</nombre>
       <activadorV id="av1.1" ventana="v2">
        <nombre>Interfaz 1</nombre>
      </activadorV>
      <activadorV id="av1.2" ventana="v3">
        <nombre>Interfaz 2</nombre>
      </activadorV>
      <activadorV id="av1.3" ventana="v5">
        <nombre>Interfaz 3</nombre>
      </activadorV>
      <activadorV id="av1.4" ventana="v9">
        <nombre>Interfaz 4</nombre>
      </activadorV>
      <activadorV id="av1.5" ventana="v11">
        <nombre>Interfaz 5</nombre>
      </activadorV>
   </ventana>

   <ventana id= "v2">
      <nombre>Ventana 1</nombre>
      <panel id= "p2.1">
        <contenidoPanel>
          <contenidoDefecto>/profesor/biografía</contenidoDefecto>
          <contenido>/profesor/biografía</contenido>
           <contenido>/profesor/españa</contenido>
           <contenido>/profesor/estudios</contenido>
           <contenido>/profesor/docencia</contenido>
           <contenido>/profesor/formulario</contenido>
           <contenido>/profesor/resultado</contenido>
           <contenidoGrupo>/profesor/localidades</contenidoGrupo>
           <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
         </contenidoPanel>
        <destinoEnlaces panel="p2.1"/>
      </panel>
      <activadorV id="av2.1" ventana="v2">
         <nombre>A biografía</nombre>
      </activadorV>
   </ventana>

<ventana id= "v3">
      <nombre>Ventana 2</nombre>

      <panel id= "p3.1" tam= "155, 265">
        <contenidoPanel>
          <contenidoDefecto>/profesor/índice</contenidoDefecto>
        </contenidoPanel>
        <destinoEnlaces panel="p4.1"/>
      </panel>

      <panel id="p2.2" tam="400, 265">
        <contenidoPanel>
           <contenidoDefecto>/profesor/biografía</contenidoDefecto>
           <contenido>/profesor/biografía</contenido>
        </contenidoPanel>
```

```
      <destinoEnlaces panel="p4.1"/>
     </panel>
 </ventana>

 <ventana id="v4">
   <nombre>Ventana 3</nombre>
   <panel id="p4.1" tam="445, 380">
      <contenidoPanel>
        <contenido>/profesor/españa</contenido>
         <contenido>/profesor/estudios</contenido>
         <contenido>/profesor/docencia</contenido>
         <contenido>/profesor/formulario</contenido>
         <contenidoGrupo>/profesor/localidades</contenidoGrupo>
         <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
       </contenidoPanel>
      <destinoEnlaces panel="p4.2"/>
   </panel>
   <panel id="p4.2" tam="265, 150">
     <contenidoPanel>
         <contenido>/profesor/resultado</contenido>
     </contenidoPanel>
     <destinoEnlaces panel="p4.2"/>
   </panel>

    <activadorV id="av4.1" ventana="v3">
     <nombre>A ventana 2</nombre>
   </activadorV>
</ventana>

<ventana id="v5">
   <nombre>Ventana 4</nombre>
   <panel id="p5.1" tam= "155, 265">
      <contenidoPanel>
         <contenidoDefecto>/profesor/índice</contenidoDefecto>
      </contenidoPanel>
      <destinoEnlaces panel="p6.1"/>
   </panel>
   <panel id="p5.2" tam="400, 265">
      <contenidoPanel>
         <contenidoDefecto>/profesor/biografía</contenidoDefecto>
      </contenidoPanel>
      <destinoEnlaces panel="p6.1"/>
   </panel>
</ventana>

<ventana id="v6">
  <nombre>Ventana 5</nombre>
  <panel id="p6.1">
    <contenidoPanel>
      <contenido>/profesor/españa</contenido>
      <contenido>/profesor/estudios</contenido>
      <contenido>/profesor/docencia</contenido>
      <contenido>/profesor/formulario</contenido>
    </contenidoPanel>
    <destinoEnlaces panel="p7.1"/>
  </panel>
</ventana>

<ventana id="v7">
  <nombre>Ventana 6</nombre>
  <panel id="p7.1">
     <contenidoPanel>
      <contenido>/profesor/resultado</contenido>
      <contenidoGrupo>/profesor/localidades</contenidoGrupo>
      <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
    </contenidoPanel>
    <destinoEnlaces panel= "p8.1"/>
  </panel>
</ventana>

<ventana id="v8">
   <nombre>Ventana 7</nombre>
   <panel id="p8.1">
     <contenidoPanel>
```

```
      </contenidoPanel>
    </panel>
    <activadorV id="av8.1" ventana= "v5">
      <nombre>A ventana 4</nombre>
    </activadorV>
</ventana>

<ventana id="v9">
    <nombre>Ventana 8</nombre>
    <panel id="p9.1" tam= "155, 265">
       <contenidoPanel>
          <contenidoDefecto>/profesor/índice</contenidoDefecto>
       </contenidoPanel>
       <destinoEnlaces panel="p9.3"/>
    </panel>
    <panel id="p9.2" tam="400, 265">
       <contenidoPanel>
          <contenidoDefecto>/profesor/biografía</contenidoDefecto>
       </contenidoPanel>
       <destinoEnlaces panel="p9.3"/>
    </panel>
    <panel id="p9.3" tam="400, 265">
     <contenidoPanel>
       <contenido>/profesor/españa</contenido>
       <contenido>/profesor/estudios</contenido>
       <contenido>/profesor/docencia</contenido>
       <contenido>/profesor/formulario</contenido>
     </contenidoPanel>
     <destinoEnlaces panel="p10.1"/>
  </panel>
</ventana>

<ventana id="v10">
  <nombre>Ventana 9</nombre>
  <panel id="p10.1">
     <contenidoPanel>
      <contenido>/profesor/resultado</contenido>
      <contenidoGrupo>/profesor/localidades</contenidoGrupo>
      <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
    </contenidoPanel>
    <destinoEnlaces panel= "p10.2"/>
  </panel>
  <panel id="p10.2" >
     <contenidoPanel>
     </contenidoPanel>
  </panel>
  <activadorV id="av10.1" ventana= "v9">
     <nombre>A ventana 8</nombre>
   </activadorV>
</ventana>

<ventana id="v11">
    <nombre>Ventana 10</nombre>
    <panel id="p11.1"  tam= "300, 380">
       <contenidoPanel>
          <contenidoDefecto>/profesor/biografía</contenidoDefecto>
       </contenidoPanel>
       <destinoEnlaces panel="p11.2"/>
    </panel>
    <panel id="p11.2" tam="445, 380">
       <contenidoPanel>
         <contenido>/profesor/españa</contenido>
          <contenido>/profesor/estudios</contenido>
          <contenido>/profesor/docencia</contenido>
         <contenido>/profesor/formulario</contenido>
       </contenidoPanel>
     <destinoEnlaces panel="p11.3"/>
  </panel>
  <panel id="p11.3" tam="200, 380">
     <contenidoPanel>
        <contenido>/profesor/resultado</contenido>
         <contenidoGrupo>/profesor/localidades</contenidoGrupo>
         <contenidoGrupo>/profesor/asignaturas</contenidoGrupo>
     </contenidoPanel>
```

```
        <destinoEnlaces panel="p11.3"/>
    </panel>

  </ventana>

</aplicacion>
```

C.2 *Lire en Français*

C.2.1 DTD de contenidos

```
<!ELEMENT contenidos (artículo, frases, definiciones)>

<!ELEMENT artículo (ancla+)>
<!ATTLIST artículo imagen CDATA #REQUIRED>
<!ELEMENT ancla EMPTY>
<!ATTLIST ancla href CDATA #REQUIRED coords CDATA #REQUIRED>

<!ELEMENT frases (frase+)>
<!ELEMENT frase (palabra+)>
<!ATTLIST frase id ID #REQUIRED>
<!ELEMENT palabra (#PCDATA)>
<!ATTLIST palabra href IDREF #REQUIRED>

<!ELEMENT definiciones  (definición+)>
<!ELEMENT definición (pal, morfología, significado?)>
<!ATTLIST definición id ID #REQUIRED>
<!ELEMENT pal (#PCDATA)>
<!ELEMENT morfología  (#PCDATA)>
<!ELEMENT significado  (#PCDATA)>
```

C.2.2 Documento de contenidos

```
<?xml version="1.0" encoding= "ISO-8859-1"?>

<!DOCTYPE contenidos SYSTEM "cLEF.dtd">

<contenidos>
  <artículo imagen="c:\plumbingXJ\GAP\documentos\incendio.jpg">
    <ancla href="f1" coords=" 0, 0, 315, 0, 315, 55, 0, 55, 0, 0"/>
    <ancla href="f2" coords=" 0, 55, 315, 55, 315, 100, 0, 100, 0, 55"/>
  </artículo>

  <frases>
     <frase id="f1"><palabra href="une1">Une </palabra><palabra href="famille1">famille
</palabra><palabra href="decimee1">décimée </palabra><palabra href="par1">par
</palabra><palabra href="l1">l'</palabra><palabra href="incendie1">incendie
</palabra><palabra href="d1">d'</palabra><palabra href="un1">un </palabra><palabra
href="appartement1">appartement</palabra>
  </frase>

    <frase id="f2"><palabra href="une1">Une </palabra><palabra href="bougie1">bougie
</palabra><palabra href="serait1">serait </palabra><palabra
href="l1">l'</palabra><palabra href="origine1">origine </palabra><palabra href="du1">du
</palabra><palabra href="sinistre1">sinistre </palabra><palabra href="qui1">
qui</palabra> <palabra href="aCoute1">a coûté </palabra><palabra href="la1">la
</palabra><palabra href="vie1">vie </palabra><palabra href="a2">à </palabra><palabra
href="six1">six </palabra><palabra href="personnes1">personnes </palabra><palabra
href="dans1">dans </palabra><palabra href="les1">les </palabra><palabra
href="hautsDeSeine1">Hauts-de-Seine</palabra>
  </frase>

</frases>

<definiciones>
<definición id="une1">
<pal>une</pal>
<morfología>art.indet.fem.sing.</morfología>
</definición>
<definición id="famille1">
<pal>famille</pal>
<morfología>nom. fem. sing.</morfología>
<significado>Conjunto de personas ligadas entre ellas por matrimonio o por
filiación.</significado>
</definición>
```

```
<definición id="decimee1">
<pal>décimée</pal>
<morfología>part. pas. utilizado como adjetivo, fem. sing.</morfología>
<significado> Inf. décimer. Familia en la que han muerto muchas personas.
</significado>
</definición>
<definición id="par1">
<pal>par</pal>
<morfología>prep.</morfología>
<significado>Introduce la causa o el motivo.</significado>
</definición>
<definición id="l1">
<pal>l'</pal>
<morfología>art. det. masc. sing.</morfología>
<significado>Forma elidida del artículo "le".</significado>
</definición>
<definición id="incendie1">
<pal>incendie</pal>
<morfología>nom. masc. sing.</morfología>
<significado>Fuego que se propaga y provoca daños materiales y/o humanos.</significado>
</definición>
<definición id="d1">
<pal>d'</pal>
<morfología>prep.</morfología>
<significado>Forma elidida de la preposición "de".</significado>
</definición>
<definición id="un1">
<pal>un</pal>
<morfología>art. indet. masc. sing.</morfología>
</definición>
<definición id="appartement1">
<pal>appartement</pal>
<morfología>nom. masc. sing.</morfología>
<significado>Vivienda en una casa de varias plantas.</significado>
</definición>
<definición id="bougie1">
<pal>bougie</pal>
<morfología>nom. fem. sing.</morfología>
<significado>Pequeño cilindro de cera, atravesado en su parte central por un cordón, y
que sirve de mecha. Se utiliza como elemento de decoración.</significado>
</definición>
<definición id="serait1">
<pal>serait</pal>
<morfología>verb. ind. cond. pres. 3ª pers. sing.</morfología>
<significado>Inf. être. Sirve de cópula.</significado>
</definición>
<definición id="a1">
<pal>à</pal>
<morfología>prep.</morfología>
<significado>Indica el punto de partida de una situación.</significado>
</definición>
<definición id="l2">
<pal>l'</pal>
<morfología>art. det. fem. sing.</morfología>
<significado>Forma elidida del artículo "la".</significado>
</definición>
<definición id="origine1">
<pal>origine</pal>
<morfología>nom. fem. sing.</morfología>
<significado>Principio o causa de un acontecimiento.</significado>
</definición>
<definición id="du1">
<pal>du</pal>
<morfología>art. det. contracto.</morfología>
<significado>Contracción de la preposición "de" y del artículo "le".</significado>
</definición>
<definición id="sinistre1">
<pal>sinistre</pal>
<morfología>nom. masc. sing.</morfología>
<significado>Acontecimiento catastrófico que ocasiona daños o pérdidas importantes en
propiedades o personas.</significado>
</definición>
<definición id="qui1">
<pal>qui</pal>
```

```
<morfología>pron. rel. sujeto</morfología>
<significado>Sustituye a "sinistre".</significado>
</definición>
<definición id="aCoute1">
<pal>a coûté</pal>
<morfología>verb. 1ª conjug. ind. pret. perf. 3ª pers. sing.</morfología>
<significado>Inf. coûter. "A coûté la vie à"  Causar la muerte a alguien.</significado>
</definición>
<definición id="la1">
<pal>la</pal>
<morfología>art. det. fem. sing.</morfología>
</definición>
<definición id="vie1">
<pal>vie</pal>
<morfología>nom. fem. sing.</morfología>
<significado>Transcurso de tiempo que va desde el nacimiento hasta la
muerte.</significado>
</definición>
<definición id="a2">
<pal>à</pal>
<morfología>prep.</morfología>
<significado>Introduce a "six personnes".</significado>
</definición>
<definición id="six1">
<pal>six</pal>
<morfología>adj. num. card.</morfología>
<significado>Número seis.</significado>
</definición>
<definición id="personnes1">
<pal>personnes</pal>
<morfología>nom. fem. plur.</morfología>
<significado>Seres humanos.</significado>
</definición>
<definición id="dans1">
<pal>dans</pal>
<morfología>prep.</morfología>
<significado>Indica el lugar.</significado>
</definición>
<definición id="les1">
<pal>les</pal>
<morfología>art. det. masc. plur.</morfología>
</definición>
<definición id="hautsDeSeine1">
<pal>Hauts-de-Seine</pal>
<morfología>nom. prop. lug.</morfología>
<significado>Departamento.</significado>
</definición>
</definiciones>
</contenidos>
```

C.2.3 Documento de la aplicación

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="ap4">

   <ventana id= "v1">
      <nombre>ventana 1</nombre>

      <panel id= "p1.1" tam="350, 405">
        <contenidoPanel>
         <contenidoDefecto>/contenidos/artículo</contenidoDefecto>
        </contenidoPanel>
          <destinoEnlaces panel="p1.2"/>
      </panel>

      <panel id= "p1.2" tam="350, 100">
        <contenidoPanel>
         <contenidoGrupo>/contenidos/frases</contenidoGrupo>
        </contenidoPanel>
```

```
         <destinoEnlaces panel="p1.3"/>
      </panel>

      <panel id= "p1.3" tam="200, 200">
        <contenidoPanel>
         <contenidoGrupo>/contenidos/definiciones</contenidoGrupo>
        </contenidoPanel>
      </panel>
   </ventana>

</aplicacion>
```

C.3 Tutoriales XML

C.3.1 DTD de contenidos

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT contenidos (índice, tema+)>

<!ELEMENT índice (ancla+)>
<!ATTLIST índice imagen CDATA #REQUIRED>
<!ELEMENT ancla EMPTY>
<!ATTLIST ancla  href CDATA #REQUIRED
                 coords CDATA #REQUIRED>

<!ELEMENT tema (subÍndice, contenido)*>
<!ELEMENT subÍndice (#PCDATA|entrada)*>
<!ATTLIST subÍndice  id ID #REQUIRED>
<!ELEMENT entrada (#PCDATA)>
<!ATTLIST entrada href CDATA #REQUIRED>
<!ELEMENT contenido (#PCDATA|destino)*>
<!ATTLIST contenido id ID #REQUIRED>
<!ELEMENT destino (#PCDATA)>
<!ATTLIST destino name ID #REQUIRED>
```

C.3.2 Documento de contenidos

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE contenidos SYSTEM "cXML.dtd">

<contenidos>
  <índice imagen="c:\plumbingXJ\GAP\documentos\indiceXML.jpg">
      <ancla href="SIxml" coords="0, 0, 92, 0, 92, 36, 0, 36, 0, 0"/>
      <ancla href="SIsax" coords="0, 40, 92, 40, 92, 76, 0, 76, 0, 40"/>
      <ancla href="SIdom" coords="0, 80, 92, 80, 92, 116, 0, 116, 0, 80"/>
      <ancla href="SIxslt" coords="0, 120, 92, 120, 92, 156, 0, 156, 0, 120"/>
      <ancla href="SIxhtml" coords="0, 160, 92, 160, 92, 196, 0, 196, 0, 160"/>
  </índice>

<tema >
<subÍndice id="SIxml">
INTRODUCCIÓN A XML

Contenido:
<entrada href="xml">1.- Que es XML: </entrada>
<entrada href="xml#ejXML">2.- Algunos ejemplos de lenguajes basados en XML: </entrada>
<entrada href="xml#fbXML">3.- Fundamentos básicos: </entrada>
<entrada href="xml#hXML">4.- Herramientas XML: </entrada>
</subÍndice>
<contenido id="xml">
INTRODUCCIÓN A XML
<![CDATA[
1.- Que es XML:

1.1.- Origen de los lenguajes de marcado:
· El origen de XML se encuentra en el "estándar de lenguaje de marcado generalizado"(
SGML )(Standard Generalized Markup Language), de hecho XML puede considerarse un
subconjunto de SGML.
·SGMLes un metalenguaje de marcado, es decir, es un lenguaje que sirve para definir
lenguajes de marcado específicos. Estos lenguajes definidos mediante SGML pueden
adaptarse a un sistema de tratamiento de documentos, o a un sistema de procesamiento de
texto concretos. La principal característica de los lenguajes de marcado generalizados
es la de utilizar marcas descriptivas, es decir, marcas que ponen de manifiesto la
estructura lógica de un texto o documento. En contraposición están los lenguajes de
marcado que utilizan marcas imperativas, esto es, marcas que implican ordenes al sistema
que debe procesar el texto. En este contexto XML es el fruto del esfuerzo del W3 por
simplificar SGML y adaptarlo a las necesidades de la "Web".
Sistemas de procesamiento de texto : sistemas de computadora capaces de automatizar
total o parcialmente el proceso de creación de un documento de texto y el de edición
```

- Las acciones básicas en la edición electrónica de un texto: tecleamos un documento, e indicamos, mediante marcas especiales, como queremos formatearlo. La computadora procesa esta información e imprime (o presenta en pantalla) el texto de acuerdo con el formato indicado. Durante el proceso de edición electrónica, un texto es la combinación de la información que constituye el contenido original del documento y de las marcas que indican el formato (otros ejemplos de sistemas de marcas son RTF y LaTeX). Los procesadores de texto más modernos facilitan la edición del documento mediante los sistemas llamados WYSIWYG "lo que usted ve (durante la edición) es el resultado que usted obtendrá (al imprimir)" (p.e. Microsoft Word y Wordperfect). Antes de los sistemas WYSIWYG las marcas se añadían al texto tecleandolas directamente, p.e. cuando creábamos un texto en LaTeX en un entorno UNIX:

Este es un ejemplo de texto marcado. Contiene palabras que son {\italicized} (cursiva), {\bold faced} (negrita), {\small small} (minúscula) y {\large large} (mayúscula).(1)

- Marcado generalizado : Con la generalización del uso de los sistemas informáticos para procesar información, se empieza a exigir más de los sistemas de almacenamiento, búsqueda, gestión y edición de documentos. Así, comienzan a utilizarse marcas con un propósito más amplio que el simple formato (p.e. para la búsqueda selectiva de información). Esto plantea un problema, cada sistema utiliza un lenguaje de marcado independiente lo que hace que no puedan comunicarse fácilmente entre si. La solución es evidentemente crear un lenguaje estándar. Características que debería tener un lenguaje de marcas estándar:

·debe ser general. Este lenguaje será utilizado por sistemas diferentes, cada uno con un ámbito de aplicación distinto, luego el mismo lenguaje de marcas debe valer para todos los sistemas;

· debe ser flexible: debe poder adaptarse a cualquier ámbito de aplicación;

· debe ser robusto: los sistemas deben trabajar con los documentos de forma fiable, (el marcado tiene que estar basado en normas claras y precisas).

Así surgió SGML .

El lenguaje HTML y la Web: En 1989, Tim-Barners-Lee, entonces en el CERN (Centro Europeo de Investigación Nuclear), quiere utilizar documentos de texto con hiperenlaces para faciltar la difusión de la información científica que producen distintos centros de investigación. Su colega Anders Berglund le aconseja usar SGML. A partir de un ejemplo de tipo de documento escrito en SGML desarrollan una primera versión de un lenguaje para difundir documentos con hiperenlaces: el lenguaje de marcado de hipertexto (HTML).

· Características de HTML inducidas por SGML:

[A] - salvo excepciones, los tipos de elemento de HTML son descriptivos (no del tipo de los del ejemplo (1): en un documento HTML, decimos que se tiene que colocar una tabla: <TABLE/>; y no como debe hacerse la tabla), con esto se consigue una cierta independencia de la aplicación final, (la tabla se puede mostrar en una consola de texto, en un interfaz gráfico o en un interfaz para invidentes);

- HTML usa la notación simple de marcado mediante etiquetas descriptivas de SGML, lo que implica que pueden crearse y leerse documentos HTML con cualquier editor de texto.

· Características propias de HTML:

[B] - HTML usa una conjunto fijo de etiquetas: no es extensible. Es decir, HTML es un lenguaje, no un metalenguaje con el que podamos definir otros lenguajes adaptados a nuestras necesidades (como ocurre con SGML)

· De [A] y [B], podríamos decir que HTML es un lenguaje útil para describir la presentación de documentos en términos de título, párrafo, tablas, etc., pero no es adaptable, o fácilmente extensible para resolver otro tipo de requisitos.

· Como consecuencia de [B], la rigidez de HTML se convirtió en un problema, y para subsanarla, los vendedores empezaron a incluir en sus navegadores extensiones no compatibles con HTML y así ganar cuota de mercado. El resultado era que se estaba perdiendo la normalización!!!

· La W3 , para atacar este problema, planteó 3 ámbitos de actuación:

- adoptar una solución similar a la adoptada en SGML para separar la descripcón del contenido de un documento de la descripción de su forma de presentación. El resultado fueron las hojas de estilos en cascada (CSS). La característica similar de SGML se llama LPD "enlaces de definición de procesos"

- también se ideó un mecanismo para añadir abstracción a HTML (clases e identificadores para elementos)

(esto reconducía la situación a algo que satisface la 3 reglas que comentamos antes; de nuevo HTML era un estándar, y permitía algunas extensiones o adaptabilidad al campo de aplicación).

No obstante, el frágil HTML extensible y CSS sólo tienen un carácter provisional; pronto se hace evidente la necesidad de incorporar una característica fundamental de SGML: la capacidad de definir tipos de documentos y la posibilidad de poder comprobar el tipo al que pertenece un documento concreto.

- la W3 decidió desarrollar un metalenguaje que fuese un subconjunto de SGML y que recogiese junto con las principales ventajas de SGML la sencillez de HTML; este lenguaje se llamó XML , lenguaje de marcado extensible (eXtensible Markup Language).

Posteriormente este lenguaje ha servido para definir muchos otros lenguajes específicos, p.e. el lenguaje de enlaces extensible XLINK y el de estilos XSL.

1.2.- Relación entre SGML-XML-HTML:

```
- SGML : es un metalenguaje de marcado general y extensible; esto significa que con él
podemos describir/definir cualquier otro lenguaje de marcas (p.e. HTML).
- XML : adopta las principales características (deja fuera las más complejas y las menos
útiles) de SGML, (es "casi" un "subconjunto" de SGML), pero sigue siendo general y
extensible, (con XML podemos seguir definiendo lenguajes de marcas concretos; p.e.
HTML).
- HTML : es un lenguaje de marcado concreto, (sería un elemento del conjunto de
lenguajes definibles con SGML o XML).
]]>
<destino name="ejXML">2.- Algunos ejemplos de lenguajes basados en XML:</destino>

2.1.- Posibles aplicaciones de XML:
. XML tiene futuro: empresas, con una cuota tan importante de mercado, como Microsoft,
IBM, Oracle, Hitachi y Sun, están realizando un esfuerzo importante desarrollando nuevos
entornos y herramientas basadas en XML.
. XML se está utilizando o se utilizará en un futuro cercano en los entornos de:
- bases de datos,
- comercio electrónico,
- intercambio de información científica,
- automatización de contenidos Web,
...
(En las siguientes secciones veremos algún ejemplo de ello).

2.2.- Hitachi semiconductors:
- Hitachi semiconductors mantiene cerca de 1.000 hojas de datos (fundamentalmente con
características de los chips y especificaciones de su interfaz), con unas 70.000
paginas, a las que acceden clientes potenciales, e ingenieros de diseño; a través de
papel, CD-ROM y Web.
- La empresa necesitaba un método para simplificar el control de versiones y poder
publicar en todos los medios a partir de una sola fuente sencilla y de fácil gestión.
- Para resolver el problema, se utilizó el software "FrameMaker+SGML" de Adobe (una
herramienta integrada de escritura y edición basada en SGML); en una primera fase la
empresa adoptó el productó para crear un solo archivo fuente para varios formatos de
salida, como SGML, HTML, PDF y RTF. En una segunda fase se incluyó XML.
- Con ello la empresa logró reducir en un 40 por ciento sus gastos de impresión, reducir
el ciclo de impresión de 3 meses a 1 mes, (los clientes se descargan las hojas de datos
en formato PDF); se evitan también bastantes gastos de envio,...

2.3.- Washington Post:
- La sección de empleo del Washington Post en su versión digital, creció de forma
rápida: alcanzando la cifra de 25.000 ofertas de empleo, obtenidas de 150 fuentes
distintas. El problema con el que se encontraban era manejar esa gran cantidad de
información que además venía de fuentes dispares y con formato distinto.
- Se introdujo XML como solución: agregando un nivel entre la fuente de la oferta y su
representación final. Esto permitió que todos los datos se tratasen de igual forma,
independientemente de su origen.
- El sitio de ofertas del Washington Post es un ejemplo claro de la agregación de datos
de nivel intermedio basada en XML. (También hace uso de la tecnología VDB , -que
presenta Internet a los clientes como si fuese una base de datos virtual-).

2.4.- Jungle shopping guide:
- " Jungle shopping guide " es un servicio para ir de tiendas que utiliza la tecnología
VDB (bases de datos virtuales), para agregar decenas de sitios Web a una gran cantidad
de categorías de productos, constituyendo de esa forma una guía única de compras.
- Shopping Guide incrementa la capacidad de XML para entregar resultados, ya que puede
ser manejado por el navegador sin tener que dar rodeos hasta el servidor Web.

<destino name="fbXML">3.- Fundamentos básicos:</destino>
<![CDATA[
3.1.- Contenido: elementos-atributos:
· Los documentos XML están formados por elementos ; la información que puede contener
cada elemento se define en la DTD del documento (la gramática que describe de forma
precisa el tipo de documento) .
· Los elementos pueden dividirse en:
- elementos con contenido: incluirán una etiqueta de inicio y una etiqueta de fin, que
delimitan el contenido del elemento; la etiqueta de inicio sería el nombre del elemento
entre ángulos (<...>) y la de fin tendría además un barra inclinada (</...>).
Ejemplo: <título>Esto es un título.</título>
- elementos sin contenido (o vacíos): (normalmente serán elementos con atributos
asociados), se pueden incluir con la notación anterior, pero sin especificar contenido:
o usando una etiqueta de elemento vacío que no requiere ni contenido ni etiqueta de fin
de elemento:
<ElementoVacio/>.
```

· Los elementos, además de contenido pueden tener atributos. Los atributos sirven para recoger características o propiedades de los elementos. Los atributos tendrán un nombre que los identifica dentro del elemento, y un valor asociado en cada ocurrencia de un elemento concreto. Los atributos que tiene un elemento, y los tipos de valores que pueden tomar, se especifican dentro de la DTD del documento.
Ejemplo de elemento con atributos:
<Persona altura="1.72" peso="70">Mariano</Persona>
Los valores de los atributos deben ir entrecomillados (con comillas simples ', o con comillas dobles ", - cuando se utiliza un tipo de comillas, el otro puede utilizarse dentro del valor del atributo -).

3.2.- Prólogo-cabecera:
· Los documentos de XML deben empezar con un prólogo que describa la versión de XML utilizada, el tipo de documento y otras características; el prólogo se compondrá de:
- una declaración XML , que incluye: información de versión (1), declaración de codificación (2) (que indica el tipo de codificación de caracteres utilizado), y una declaración de documento autónomo (3) (en ella se indica que componentes de la declaración de tipo de documento son necesarios para completar el procesamiento del documento).
Ejemplos:
· Declaración mínima XML:
<?xml version="1.0"?> .
· Declaración XML más amplia:
<?xml version="1.0"(1) encoding="UTF-8"(2) standalone="yes"(3) ?>
- una declaración de tipo de documento: (DTD) que es la base del concepto de validez estructural de un documento: en la DTD se describen los elementos disponibles, se imponen restricciones en la coincidencia y contenido de los elementos, ... así se consigue un sistema de información mas robusto (se entiende que todos los documentos que compartan una DTD serán consistentes).
· Los documentos que satisfagan su DTD se dirá que son válidos (en contraposición a los que únicamente satisfagan la sintaxis básica de XML, que se llamarán bien formados), de esta forma un documento puede:
· estar mal formado (ni siquiera es válido desde el punto de vista de la sintaxis de XML); (ej. <Persona>Hombre),
· estar bien formado, pero no ser válido (es sintácticamente válido desde el punto de vista de XML, pero no satisface las restricciones impuestas por su DTD), también podría estar bien formado y no tener asociada ninguna DTD;
· estar bien formado y ser válido.
(un documento nunca podría ser válido y estar mal formado).
· Hay dos tipos de DTD :
- internas : la DTD se incluye en el propio documento. La declaración de tipo de documento se introduce con "<! DOCTYPE ", a continuación el nombre del elemento raíz del documento, después las declaraciones de tipo de elemento entre corchetes "[...]" y se cierra la declaración con ">".
ejemplo: <!DOCTYPE Clase[
<!ELEMENT Clase (Profesor,Alumno+)>
<!ELEMENT Profesor (Nombre, Asignatura)>
<!ELEMENT Alumno (Nombre)>>]>
- externas : la DTD está en otro fichero, (es mas útil de esta forma, así múltiples documentos pueden compartir la definición de su tipo); en este caso tendremos: "<!DOCTYPE" seguido del elemento raíz del documento, después "SYSTEM", el nombre ó URL del fichero que contiene la DTD, y por último ">".
ejemplo: <!DOCTYPE Clase SYSTEM "http://www.pepito.com/Clase.dtd">

3.3.- Declaración de tipo de elemento:
· Las declaraciones de tipo de elemento son partes de la DTD en las que se especifica que contenido van a poder tener los elementos dentro del documento.
· Las declaraciones de elemento empezarán con "<!ELEMENT, seguidas del nombre del elemento, una especificación de contenido, y acaban con ">".
· En cuanto a las especificaciones de contenido , hay 4 posibilidades:
- contenido EMPTY : el elemento será siempre vacío, (se usará cuando sólo interesan los atributos del elemento),
ejemplo: <!ELEMENT ElementoVacio EMPTY>
- contenido ANY : dentro del elemento podemos tener cualquier cosa, (se suele utilizar muy pocas veces, porque hace que todo su contenido esté desectructurado).
ejemplo: <!ELEMENT CualquierCosa ANY>
- contenido " mixed ": cuando el elemento puede contener caracteres, ó una mezcla de caracteres y subelementos, (para especificar que elementos puede contener y en que orden se utiliza un modelo de contenido).
ejemplo: <!ELEMENT SoloCaracteres (#PCDATA)>
ejemplo: <!ELEMENT MezclaCaracteresSubElementos (#PCDATA | SubElemento)*>
- contenido " element ": cuando sólo se introducen subelementos en la especificación de contenido.
ejemplo: <!ELEMENT Alumno (Nombre, Apellidos, Curso)>

· Para los elementos con contenido mixed y element, la especificación de que subelementos pueden contener y en que orden, se realiza mediante un modelo de contenido . El modelo de contenido sigue al nombre del elemento en la declaración de elemento, y va encerrada entre paréntesis, los subelementos podrán ir separados por:
- comas (","): indica secuencia de subelementos , de tal forma que si tenemos:
<!ELEMENT Alumno (Nombre, Apellidos, Curso)>
estamos especificando que el elemento Alumno estará formado por la secuencia de 3 elementos (Nombre, Apellidos y Curso), que además deberán aparecer en ese orden;
- barras ("|"): indica opción de subelementos , de tal forma que si tenemos:
<!ELEMENT TecnicoInformatica (Gestion | Sistemas)>
decimos que dentro de la marca: "TecnicoInformatica" puede aparecer o el subelemento "Gestion" o el subelemento "Sistemas", (debe aparecer uno y sólo uno de los dos).
También es posible combinar opciones y secuencias usando paréntesis:
ejemplo: <!ELEMENT TecnicoInformatica ((Gestion | Sistemas), FechaFinalizacion)>, (ahora el elemento "TecnicoInformatica" contendrá dos subelementos: primero un elemento "Gestion" o un elemento "Sistemas", y un segundo subelemento: "FechaFinalizacion").
· También es posible usar en los modelos de contenido indicadores de frecuencia para subelementos o grupos de subelementos:
- "?" : opcional (0 ó 1 vez),
- "*" : opcional y repetible (0 ó mas veces),
- "+" : necesario y repetible (1 ó mas veces).
ejemplo: <!ELEMENT TecnicoInformatica ((Gestion | Sistemas), FechaFinalizacion?)> (ahora es posible que no tengamos el subelemento FechaFinalizacion).
ejemplo: <!ELEMENT Texto (Titulo , Capitulos+)> (tenemos un elemento texto con un título y 1 ó mas capítulos).
ejemplo: <!ELEMENT Encuesta (Presentacion, (Pregunta, Respuesta)+, Resultado?)>(el elemento encuesta estaría formado por una presentación, seguida de una secuencia de 1 ó mas pregunta-respuesta, y quizas un resultado).

3.4.- Declaración de lista de atributos:
· A la hora de decidirse entre incluir un atributo o un subelemento , tenemos que tener en cuenta las diferencias entre ambos:
- los atributos no pueden contener elementos, y no existen "subatributos";
- los atributos sólo recibirán valor una vez para un elemento, y el orden en el que se especifiquen resulta irrelevante.
· Los atributos se especifican para un determinado tipo de elemento usando una declaración de lista de atributos: empieza con "<!ATTLIST ", luego el nombre de un tipo de elemento, y después una lista de declaración de atributos, cada una de las cuales llevará el nombre del atributo, su tipo y su valorpor defecto. La declaración acaba con ">".
ejemplo: <!ATTLIST TarjetaDeVisita Nombre CDATA #REQUIRED
Direccion CDATA #REQUIRED
Telefono CDATA #REQUIRED
EMail CDATA #IMPLIED>
· Tipos de atributos :
- CDATA : ("character data"), cualquier cadena de caracteres;
- NMTOKEN : cadena de caracteres, con las restricciones típicas de los identificadores (secuencia de letras y dígitos más algún otro carácter "." "-" "_" ":", que empieza por letra);
- atributos enumerados : atributos que sólo pueden tomar valores de una lista dada,
ejemplo: <!ATTLIST Informatico NivelesInteligencia (MuyListo, Listo, Tonto, MuyTonto, Mariano) #REQUIRED>
- atributos de anotación : atributos que permiten declarar que el contenido de un elemento se ajusta a una determinada anotación declarada,
ejemplo: (modos de presentar fechas). <!ATTLIST Fecha NOTATION (EUROPEAN-DATE | USA-DATE | ISO-DATE) #REQUIRED> (las notaciones deben haberse declarado previamente).
- atributos ID : (identificador de elemento), indica que ese atributo "es una clave", lo que significa que no podremos tener dos elementos en un documento con el mismo valor en un atributo ID;
- atributos IDREF : (referencia a identificador de elemento), indica que ese atributo es "referencia a una clave": el valor que contenga ese atributo deberá ser igual al de algún atributo ID de algún elemento que esté en el documento;
- atributos ENTITY : suelen utilizarse para hacer referencia a objetos externos que no deberán analizarse sintácticamente en XML (imagenes, videos, ...).
· Valores por defecto :
- se incluyen después de la especificación de tipo de atributo, entre comillas dobles:
ejemplo: <!ATTLIST Zapatos Talla CDATA "40">
ejemplo: <!ATTLIST Informatico NivelesInteligencia (MuyListo, Listo, Tonto, MuyTonto, Mariano) "Listo">
- hay veces en las que se puede permitir que el valor de un atributo quede sin especificar, y sin darle un valor por defecto, (por ejemplo hay camisetas que no tienen una talla determinada, son talla única); en este caso se especificará que el atributo es IMPLIED .
ejemplo: <!ATTLIST Camiseta Talla CDATA #IMPLIED>

(para atributos del tipo anterior, el usuario no está obligado a darles valor, sería correcto poner: <Camiseta>...</Camiseta>).
- la situación contraria a la anterior se da cuando queremos que al atributo se le de un valor, en ese caso se especificará como REQUIRED el atributo.
ejemplo: <!ATTLIST Imagen URL CDATA #REQUIRED>

3.5.- Entidades
- Las entidades permiten dividir el documento en varios ficheros, son muy útiles para mantener y reutilizar fragmentos de documentos.
- Las entidades pueden verse como abreviaturas (en la que la abreviatura sería la entidad y la forma larga sería el contenido de la entidad - que puede ser desde un solo carácter, hasta un texto entero -).
- También pueden verse como una caja con una etiqueta: la etiqueta sería el nombre de la entidad, el contenido de la caja es algun tipo de texto o dato. La declaración de la entidad crea el cuadro y le asocia la etiqueta. Y luego en el resto del documento podemos usar la etiqueta para referirnos a toda la caja, (a su contenido).
- La declaración de las entidades se lleva a cabo en la declaración del tipo de documento, para ello se incluye primero "<!ENTITY", después el nombre de la entidad, despues entre comillas el contenido de la entidad y se acaba con ">". (Esta sintaxis tiene algunas variaciones segun el tipo de entidad, como ya veremos).
Ejemplo: <!ENTITY DTD "Definicion de tipo de documento">.
- Las referencias a entidades se realizan con "&" y el nombre de la entidad; o con "%" y el nombre de la entidad (si es una entidad parametro).
- Tipos de entidades :
. En funcion de su situación física:
- pueden ser internas : el contenido de la entidad se encuentra definido en la propia declaración de la entidad (el ejemplo anterior sería un una entidad interna);
- pueden ser externas : el contenido de la entidad esta en otro fichero distinto al del documento; la especificación de este tipo de documento se hace con: "<!ENTITY", el nombre de la entidad, "SYSTEM", y el nombre del fichero (su URL) entre comillas, se acaba la declaración con ">".
. En función de su contenido:
- pueden ser no analizadas: el contenido al que hace referencia la entidad no es contenido XML, y por lo tanto no debe analizarse, (p.e. imagenes, videos, ...); hay veces en las que tambien resulta útil incluir contenido XML como entidades externas no analizadas (p.e. si tenemos un tipo de documento "ejemplo", y queremos incluir estos documentos en otros documentos: "libro",en lugar de extender el tipo de documento con etiquetas para ejemplo, lo podríamos incluir como entidad externa no analizada).
- pueden ser analizadas : cuando el contenido de la entidad es texto XML que debe analizarse;
ejemplo: <!ENTITY Titulo "<Title>Capitulo 1</Title>">]
. En función del lugar donde vayan a utilizarse:
- pueden ser generales : si pueden referenciarse en cualquier parte del documento,(tanto en la cabecera como en el cuerpo del documento XML),
- pueden ser parámetro : si solo pueden referenciarse en la cabecera del documento, (el contenido de la entidad será una - o un conjunto de-, declaración de entidad, declaración de lista de atributos, ..., - esto es útil para reutilizar partes de la declaraciones en varios tipos de documentos -); la declaración de este tipo de entidades se hace con: "<!ENTITY", seguido de "%", y el resto igual que siempre; las referencias a este tipo de entidades se hacen con "%" mas el nombre de la entidad,
ejemplo: <!ENTITY % EntidadEjemplo "<!ELEMENT Elemento (#PCDATA)">
ejemplo: <!ENTITY % EntidadEjemplo SYSTEM "OtraDTD.dtd">
]]>
<destino name="hXML">4.- Herramientas XML:</destino>

4.1.- Algunas herramientas XML:
. Hay una gran variedad de herramientas relacionadas con XML, que suelen estar especializadas en alguna tarea en concreto:
- editores: permiten la creación de documentos xml, dtds ,... algunos son visuales y permiten trabajar sobre documentos marcados a personas que no conocen bien la sintaxis del lenguaje;
- parsers: analizadores de documentos de la familia XML, los hay que siguen distintas filosofías (DOM-SAX), según nos interese la flexibilidad o la eficiencia;
- "transformadores": permiten la transformación de documentos XML a otros documentos (XML o en cualquier otro formato);
- herramientas para obtener información estructurada a partir de información no estructurada;
- navegadores: actualmente Internet Explorer (desde su version 4), soporta ya XML y es de esperar próximamente otros navegadores también lo hagan.
Y muchas mas...
</contenido>
</tema>

<tema>

```
<subÍndice id="SIsax">
APUNTES SOBRE SAX

Contenido:
<entrada href="sax">1.- Introducción </entrada>
<entrada href="sax#pSAX">2.- El Parser </entrada>
</subÍndice>
<contenido id="sax">
APUNTES SOBRE SAX

1.- Introducción

1.1.- Definición
SAX .- (Serial Acces to XML) Mecanismo, guiado por eventos, de acceso serie para
procesar documentos XML .

1.2.- El parser y sus componentes
Los parsers que siguen el modelo SAX son una aplicación que recorre el documento XML una
sola vez y, en base a lo que va encontrando en él lanza excepciones del tipo
correspondiente.
El objeto "parser" tiene como atributos los objetos que se encargan de manejar dichas
excepciones.
Cada "parser" tiene sus propios manejadores de eventos por defecto. Nosotros podremos
pisarlos, si queremos, con nuestras clases.

1.3.- Pros y contras
Ventajas.- Es el mejor protocolo para utilizar en servidores y programas orientados a
red, porque es más rápido y usa menos memoria que DOM.
Desventajas.- El único acceso que permite es el serie, con lo que no se puede tener un
acceso hacia atrás en el árbol del documento. Esto añade más programación a nuestras
aplicaciones que, sumado al hecho de que la programación guiada por eventos es menos
intuitiva, hace que la creación y mantenimiento de dichas aplicaciones sea más complejo.

1.4.- Tipos de "parsers" SAX
- Validante .- Valida el documento comparándolo con una DTD, que es necesaria para que
funcione el "parser". - No-Validante .- No se preocupa de validar. No necesita una DTD
para funcionar, pero si existe, entonces la tiene en cuenta para saber cuándo un espacio
en blanco en el Documento XML está previsto como texto y cuándo es ignorable. En caso de
no tener DTD, considera que nunca es ignorable.

<destino name="pSAX">2.- El Parser</destino>

2.1.- Componentes del Parser
El "parser" que se crea contiene diversos manejadores ( handler ) de eventos:
DocumentHandler .- El que vamos a utilizar en nuestro ejemplo. Contempla los eventos
directamente relacionados con el contenido del documento XML.
DTDHandler .- Recibe información sobre los eventos relacionados con las DTD.
ErrorHandler .- Interfaz básico para manejar los distintos errores que se pueden dar
(fatal errors, non-fatal errors, warnings).
Cada uno de ellos maneja un subconjunto de todas las excepciones que puede lanzar el
"parser". Cada excepción es contemplada en un "handler", y sólo en uno.
HandlerBase .- Es una superclase de todos los "handlers". Tiene una codificación por
defecto para manejar cada una de las excepciones posibles.
La clase que nosotros codificaremos, será subclase de ella. Así podremos redefinir los
métodos que nos interese, y dejar por defecto el resto.

2.2.- La función "main"
En cualquier aplicación que maneje un "parser" SAX, la función "main" ha de seguir los
siguientes pasos:
Habilitar la fuente de entrada del documento XML.
Crear un ejemplar de la clase Parser .
Asignar a éste un "parser" con el método ParserFactory . makeparser (aquí se dice si es
Validante o No-Validante).
Asignar al "parser" nuestros manejadores de eventos. Sólo los que nos interese
implementar.
Usa el método parse de nuestro "parser" sobre la fuente de entrada.
El resto de código está dentro de nuestros manejadores de eventos.
</contenido>
</tema>

<tema>
<subÍndice id="SIdom">
INTRODUCCIÓN A DOM.
```

```
Contenido:
<entrada href="dom">1.- XML y DOM. </entrada>
<entrada href="dom#fDOM1">2.- Fundamentos de DOM: interfaces básicos de la W3.
</entrada>
<entrada href="dom#fDOM2">3.- Fundamentos de DOM: interfaces extendidos de la W3.
</entrada>
<entrada href="dom#eDOM">4.- Ejemplo de parser DOM, (Oracle). </entrada>
</subÍndice>
<contenido id="dom">
INTRODUCCIÓN A DOM.

1.- XML y DOM.

1.1.- Qué es DOM.
· De acuerdo con la visión de DOM, a todo documento XML bien formado le corresponde un
árbol; en este árbol jugarán el papel de nodos los elementos y sus contenidos (son nodos
los atributos, los comentarios, el texto, y las instrucciones de procesamiento). El
papel de nodo raíz lo jugará el elemento principal del documento (aquel que lo define).
· DOM son las siglas de "Document Object Model": modelo de objetos de documento; y una
representación DOM de un documento XML, no será mas que el árbol que tiene asociado,
expresado en término de objetos.
· De forma más formal (tal y como lo define la W3): DOM es un API para documentos HTML y
XML; define la estructura lógica de los documentos y la forma en la que se accede y se
manipula el documento. Con DOM se pueden construir documentos, navegar por su
estructura, y añadir, modificar, o borrar elementos y su contenido.

1.2.- Origen de DOM.
· El origen de DOM se encuentra en la búsqueda de una especificación que permitiese a
los programas funcionar en todos los navegadores.
· DHTML fue el antecesor inmediato de DOM: por eso DOM en un principio se planteó en
relación con los navegadores. Sin embargo cuando se formó un grupo de trabajo de DOM en
la W3 , entró gente de otros dominios (gente que trabajaba en editores HTML-XML), muchos
de los cuales habían trabajado antes en SGML que en XML; como resultado, DOM se ha visto
muy influido por los bosques (groves) de SGML y por el estándar HyTime.

<destino name="fDOM1">2.- Fundamentos de DOM: interfaces básicos de la W3.</destino>

2.1.- El interfaz documento (Document).
· Document , representa el documento XML o HTML entero, (se corresponde con la raíz del
árbol y da acceso a todo los nodos).
· Todos los nodos del árbol del documento (elementos, texto, atributos,...), no tienen
sentido fuera de un documento; por ello el interfaz Document recoge los métodos para
crear estos elementos; el nodo que devuelven estos métodos, lleva asociado un atributo
(ownerDocument), que asocia el nodo con el documento en el que fue creado.
· Métodos:
- createElement , createDocumentFragment , createTextNode , createComment ,
createCDATASection , createProcessingInstruction , createAttribute y
createEntityReference : todos ellos generan un nodo del tipo especificado;
- getElementsByTagName : devuelve una NodeList con los elementos etiquetados con la
cadena que se le pasa por parámetro.

2.2.- El interfaz nodo (Node).
· El interfaz Node , es el principal de los definidos en DOM, ya que la mayoría de los
interfaces expanden (son hijos) a este. Representa un único nodo en el árbol del
documento. Precisión importante: aunque este interfaz incluye métodos para tratar con
hijos del nodo, no todos los interfaces que implementan el interfaz Node van a tener
hijos (p.e. Text).
· En este interfaz se definen las constantes que identificarán los distintos tipos de
nodo, en concreto tendremos:
ELEMENT_NODE , ATTRIBUTE_NODE , TEXT_NODE , C_DATA_SECTION_NODE , ENTITY_REFERENCE_NODE
, PROCESSING_INSTRUCTION_NODE , COMMENT_NODE , DOCUMENT_NODE , DOCUMENT_TYPE_NODE ,
DOCUMENT_FRAGMENT_NODE , NOTATION_NODE .
· Métodos:
- insertBefore (Node newChild, Node refChild): inserta el nodo newChild antes del nodo
refChild (hijo del nodo que recibe el mensaje); si refChild es null, inserta el nodo al
final de la lista de nodos. Devuelve el nodo que está siendo insertado;
- replaceChild (Node newChild, Node oldChild): substituye el nodo hijo (oldChild) por el
nuevo (newChild), y devuelve el nodo viejo;
- removeChild (Node oldChild): borra el nodo hijo (oldChild);
- appendChild (Node newChild): coloca el nodo nuevo al final de la lista de nodos hijos;
- hasChildNodes : pues eso;
- cloneNode (boolean deep): clona el nodo, (si deep es cierto, también clona sus hijos
recursivamente).
```

2.3.- El interfaz lista de nodos, (NodeList).
· NodeList , proporciona la abstracción de una colección ordenada de nodos.
· Se puede acceder a los elementos de la lista con index. El primer elemento de la lista será el 0; y el último el que ocupa la posición longitud de la lista menos uno.
· Métodos:
- item (int index): devuelve el elemento de la posición index;
- length (): devuelve la longitud de la lista.

2.4.- Interfaz de NamedNodeMap.
· NamedNodeMap es el interfaz para representar colecciones de nodos a las que se accede por el nombre de los nodos. También permite el acceso a los nodos por índice.
· Métodos:
- getNamedItem (String name): devuelve un nodo cuyo nombre es el especificado;
- setNamedItem (Node item): a_ade un nodo usando su atributo nodeName;
- removeNamedItem (String name): elimina un nodo con el nombre especificado;
- item (int index): devuelve el nodo que ocupa la posición especificada;
- length (): devuelve la longitud de la lista.

2.5.- El interfaz dato de carácter, (CharacterData).
· El interfaz CharacterData extiende Node con métodos y atributos para acceder en DOM a los datos que son caracteres.
· Cuidado: ningún objeto DOM se corresponde directamente con CharacterData, aunque algunos si que son hijos de él, (p.e. Text).
· Métodos:
- substringData (long offset, long count): extrae el rango de la cadena que empieza en offset y de longitud count;
- appendData (String cad): añade al final de los datos la cadena "cad";
- insertData (long offset, String arg): inserta en los datos la cadena arg en la posición offset;
- deleteData (long offset, long count): borra de los datos la subcadena que empieza en offset, y de longitud count;
- replaceData (long offset, long count, String cad): sustituye la subcadena que empieza en offset y de longitud count por la cadena cad.

2.6.- El interfaz atributo, (attr).
· El interfaz attr representa un atributo de un objeto Element. Los valores permitidos del atributo serán los expresados en la DTD del documento.
· Los objetos Attr heredan de Node, pero no son hijos del elemento al que acompañan, en DOM no se consideran en realidad como nodos del árbol del documento, sino que los considera más como propiedades de elementos que sólo tienen sentido si acompañan a un elemento.
· Métodos:
- tiene métodos para accecer a su nombre (name) y a su valor (value).

2.7.- El interfaz elemento, (Element).
· El interfaz Element sirve (obviamente) para representar los elementos de un documento XML.
· Los elementos pueden tener atributos asociados, (el interfaz Element hereda de Node, luego puede usarse el método de Node "getAttributes" para recoger el conjunto de todos los atributos de un elemento; aunque tiene métodos para acceder al valor de sus atributos por nombre).
· Métodos:
- getAttribute (String name): obtiene el valor del atributo "name" del elemento;
- setAttribute (String name, String value): hace que el valor del atributo "name" del elemento sea "value";
- removeAttribute (String name): borra el atributo "name" del elemento, (si el atributo borrado tiene un valor por defecto, se coloca este nuevo valor);
- getAttributeNode (String name): obtiene un objeto Attr con el atributo "name" del elemento;
- setAttributeNode (Attr newAttr): añade el atributo al elemento;
- removeAttributeNode (Attr oldAttr): borra el atributo del elemento;
- getElementsByTagName (String name): devuelve una NodeList con todos los elementos que son hijos del que recibe el mensaje, y tienen el nombre: "name";

2.8.- Interfaz de texto, (Text).
. El interfaz Text , representa el contenido textual de los elementos (#PCDATA) y de los atributos (CDATA).
· Para los elementos, si no hay información marcada en su contenido, el texto estará contenido en un solo objeto de la implementación del interfaz Text. Si hay marcado, al parsearse se obtendrá una lista de nodos Elementos y Text (la lista de los hijos del elemento).
· Métodos:

- splitText (long offset): divide el contenido del nodo en dos, el contenido hasta el offset se quedará en el propio nodo, y desde el offset hasta el final se guardará en otro nodo de tipo Text que es devuelto por el método.

2.9.- Interfaz para comentarios, (Comment).
· El interfaz Comment servirá para recoger el contenido de los comentarios (todos los caracteres comprendidos entre '<!--' y '-->'.

<destino name="fDOM2">3.- Fundamentos de DOM: interfaces extendidos de la W3.</destino>

3.1.- Interfaz CDATASection.
· CDATASection : las secciones CDATA se usan para que en contenido que es texto, no sea tratado como marcas. De esta forma pueden incluirse fragmentos XML como texto no marcado.

3.2.- Interfaz de tipo de documento , (DocumentType).
· Cada documento tiene un atributo "doctype" cuyo valor será null (si estamos trabajando sin validar el documento con ninguna DTD, sólo miramos si esta bien formado); o un objeto DocumentType .
· En las últimas recomendaciones de la W3, el interfaz DocumentType sólo proporciona un interfaz a la lista de entidades que se de definen en el documento, y poco mas, (están trabajando para incluir en próximas versiones namespaces y XML-esquema).
· Como los DocumenType no permiten modificaciones en este nivel (no hay especificados métodos), recogeremos aquí sus atributos, para que se vea lo que son:
- name : el nombre de la DTD (la cadena que sigue a "DOCTYPE" en XML);
- entities : un NamedNodeMap que contiene las entidades generales (tanto externas como internas), declaradas en la DTD;
- notacions : un NamedNodeMap que contiene las notaciones declaradas en la DTD.

3.3.- Interfaz para notaciones, (Notation).
· Notation : representa una notación declarada en la DTD. Una notaticón o bien declara (por nombre), el formato de un entidad no analizada; o bien se usa para la declaración formal de los "targets" de la instrucciones de procesamiento.

3.4.- Interfaz para las entidades, (Entity)
· El interfaz Entity sirve para representar entidades (analizadas o no) de un documento XML. Cuidado: representa la entidad en si, no la declaración de la entidad.
· Los parsers XML puede optar por expandir las entidades (sustituir las referencias a entidades por su contenido), en este caso no tendremos referencias a entidades en el árbol del documento.

3.5.- Interfaz para las referncias a entidades, (EntityReference).
· Los parsers DOM insertarán objetos EntityReference dentro el árbol del documento cuando tengamos referencias a entidades en el documento origen. Hay que hacer notar que las referencias a entidades predefinidas o a caracteres se expanden automáticamente por los procesadores XML o HTML; y el resto de las entidades pueden expandirse también de forma automática por el parser.

<destino name="eDOM">4.- Ejemplo de parser DOM, (Oracle).</destino>

4.1.- Parser Oracle DOM (v2).
· El parser de Oracle -java de DOM, implementa la mayoría de los interfaces descritos anteriormente, (suele dar a las clases el nombre del interfaz, anteponiendo "XML"), aunque a algunas de estas clases se les han añadido funciones no descritas en las recomendaciones de la W3 (aunque las diferencias son mínimas).
· Vamos a centrarnos en el Parser en si, (ya que las otras clases se ajustan bastante a lo comentado arriba).
· Primero describimos la clase XMLParser que sirve como clase base para las clases DOMParser y SAXPARSER. XMLParser contiene métodos para analizar documentos XML (1.0). Esta clase no puede instanciarse (es abstracta).
· Métodos de la clase XMLParser:
- parse ([1]): se encarga de analizar el fichero fuente especificado por el parámetro [1]; este método está sobrecargado, y hay varias versiones que aceptan distintos tipos para el parámetro [1] (InputSource - una clase definida en el paquete del parser-, la URL del fichero fuente, un String que contiene la URL del fichero fuente, un InputStream asociado al fichero fuente ó un Reader asociado al fichero fuente);
- setDoctype (DTD dtd): establece la DTD respecto a la cual se realizará el análisis, (no hace falta establecerla explícitamente: si está activado el modo de análisis con validación y el fichero XML incluye una etiqueta DOCTYPE);
- setValidationMode (boolean yes): si yes vale true, establece el modo de análisis con validación; si vale false únicamente comprueba que el documento esté bien formado;
- setBaseURL (URL url): establece el URL base respecto al cual se realizarán la carga de entidades y DTDs. Este método debería usarse si se utiliza parse(InputStream) para analizar el documento;

- setPreserveWhitespace (boolean flag): establece el modo en el que se preservan los espacion en blanco;
- getValidationMode (): devuelve el modo de validación que está fijado;
- getReleaseVersion (): devuelve una cadena con la versión del Parser XML de Oracle que está siendo utilizado;
- setLocale (Locale locale): se puede usar para que la aplicación establezca en "locale" el sitio donde dejar los informes de fallos.
· Ahora nos ocupamos del parser DOM propiamente dicho, la clase DomParser , sus métodos son:
- getDocument (): devuelve el árbol DOM (un objeto XMLDocument) que obtiene una vez analizado el fichero fuente;
- getDoctype (): devuelve la DTD que se ha ligado al parser;
- parseDTD ([1], String rootName): analiza la DTD externa, obtenida del fichero indicado; este método está sobrecargado: existen varias formas de especificar el fichero fuente en [1]: mediante un objeto InputSource - esta clase está definida en el paquete del parser-, mediante un InputStream asociado al fichero fuente, mediante un Reader asociado al fichero fuente, mediante el URL del fichero fuente, o con una cadena que indique la URL del fichero);
- setErrorStream ([1]): establece el stream de salida para los errores y warnings; (también está sobrecargado);
- showWarnings (boolean yes): determina si han de mostrarse o no los warnings;
- setNodeFactory (NodeFactory factory): establece un NodeFactory distinto al que usa por defecto el parser DOM, (un NodeFactory especifica los métodos para crear varios nodos del árbol DOM durante el análisis).
</contenido>
</tema>

<tema>
<subÍndice id="SIxslt">
INTRODUCCIÓN A XSLT.

Contenido:
<entrada href="xslt">1.- Introducción </entrada>
<entrada href="xslt#fXSLT">2.- Fundamentos de XSLT. </entrada>
</subÍndice>
<contenido id="xslt">
INTRODUCCIÓN A XSLT.

1.- Introducción
<![CDATA[
1.1.- Qué es XSL.
Los documentos XSL (XML Stylesheet Language) se utilizan para transformar documentos XML, bien sea en otros documentos XML, HTML o cualquier otro formato. Se usan también para realizar presentaciones de documentos XML. Un documento XSL es un documento XML válido. Sus etiquetas (elementos) siempre tienen al comienzo el prefijo <xsl: ... Mientras que las etiquetas que no lo llevan son el resultado de la transformación XSL.
]]>
<destino name="fXSLT">2.- Fundamentos de XSLT.</destino>
<![CDATA[
2.1.- Elementos básicos
- El esqueleto inicial del documento XSL tendrá una de estas dos formas, que son equivalentes:
· <?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- Aqui van los "templates" -->
</xsl:stylesheet>
· <?xml version="1.0"?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- Aqui van los "templates" -->
</xsl:transform>
- xsl:template : Es el "template" actual a ser instanciado. Representa lo que se hará cuando se encuentre un nodo con un elemento concreto en el documento origen. Puede tener tanto el texto que dará como salida, como instrucciones XSL para seguir procesando la entrada.
· Sus atributos son:
match="[elemento que coincide]" (#REQUIRED?) Es el nombre del elemento a reconocer en el documento original. Cuando se reconce, se genera el texto y se procesan las instrucciones que contiene.
Nota: El primer elemento que se reconoce es el raíz "/".
- xsl:apply-templates : Anidado con un xsl:template, indica que deben procesarse todos los hijos del nodo que se está procesando actualmente.
Nota: La forma más concisa de ponerlo es xsl:apply-templates/>
· Sus atributos son:

select="[nombre determinante]" (#IMPLIED; por defecto, se selecionan todos los nodos: elemento, texto, comentario e instrucción de procesamiento e hijos) . Elige cuáles de entre todos los nodos hijo son procesados, según su nombre.
- xsl:value-of : Toma el valor de algo y lo copia en el documento de salida.
· Sus atributos son:
select="[nombre del objeto]" (#REQUIRED) Especifica de qué objeto se toma el valor.
Nota: Si se pone "." se está uno refiriendo al elemento reconocido.
Nota: También suele ponerse <xsl:value-of select="..."/>
Nota: El valor de cada tipo de objeto es siempre un String:
Raiz: El valor del elemento raiz.
Elemento: La concatenación de todos los datos caracteres contenidos en el elemento, incluyendo los de sus descendientes. No incluye las etiquetas.
Texto: El nodo en sí mismo.
Atributo: El valor del atributo normalizado (procesadas todas las referencias). No incluye el nombre del atributo, ni los símbolos '=','"', que hay en todo atributo.
"Namespace" El URI para el "namespace".
Instrucción de procesamiento. El valor de la instrucción sin incluir el nombre .
Comentario. El texto del comentario.
- xsl:for-each : Procesa cada elemento seleccionado con su atributo:
select="[discriminante]" (#REQUIRED). Procesa todos los hijos cuyo nombre coincida con este atributo.

2.2.- Patrones a usar en los atributos Match y Select.
- Nodo Raiz : "/"
- Nombre simple de elemento : "[nombre simple]"
- Jerarquías : Usando "/" se puede uno referir específicamente a elementos que cumplan con la jerarquía, o camino de acceso, especificado. Ej:
match="/ATOM/SYMBOL"
En esta jerarquía se puede incluir el comodín * .
- Descendientes : Usando "//" antes del nombre del elemento, se estará seleccionando cualquier descendiente del nodo actual que coincida con el nombre dado, sea cual sea su profundidad a partir de la actual.
- Identificadores : Se puede uno referir a un elemento por su id. Ej:
match="id('e47')".Se asume que el elemento al que nos referiremos tiene un atributo ID.
- Se puede uno referir a un atributo poniendo @ delante del nombre del atributo.
Se puede usar @ en las jerarquías, y combinarlo con *.
- Comentarios :select="comment()". Los paréntesis siempre van vacíos.
Nota: Trozo de código que copia en la salida todos los comentarios tal cuál:
<xsl:template match="comment()">
<xsl:comment><xsl:value-of select="."/></xsl:comment>
</xsl:template>
- Instrucciones de procesamiento : Más o menos como los comentarios. Pero puede utilizar un parámetro de tipo String, indicando el nombre de la instrucción. Si no lleva nada, se cogerá la primera instrucción que se encuentre.
Ej: match="processing-instruction('xml-stylesheet')"
- Texto: Ej: select="text()". Los paréntesis siempre vacíos. El texto, por defecto, se da como salida. Si se quiere pisar ese comportamiento, se puede usar:
<xsl:template match="text()">
</xsl:template>
"|" , se puede incluir como operador OR .
Ej: match= "ATOMMIC_NUMBER | ATOMIC_WEIGHT"
Se pueden dejar espacios si se quiere.
Si se usa también "/", éste se aplica primero.
- Testear con []:
Sirve para seleccionar un elemento si cumple las condiciones expresadas entre los corchetes. Si se pone:
nombre simple .- Se elige el elemento que tenga un hijo con ese nombre.
nombres separados por "|" .- Se elige el que cumpla alguna de las opciones. se puede usar una jerarquía con "/" y condiciones de los atributos con "@". nombre='cadena' .- Se testea si "value-of" del nombre es exactamente igual a la cadena. Nota: Se puede usar cualquier XPath.

2.3.- Expresiones para seleccionar nodos (XPath).
Con ciertas palabras clave se puede uno referir, no sólo a los descendientes, sino a otros nodos relacionados con el actual.
- ancestor : Algún antecesor del actual. Hasta la misma raíz.
- ancestor-or-self :Algún antecesor o él mismo.
- attribute : Los atributos del nodo actual.
- child : Un hijo del actual. Igual que no poner nada.
- descendant : Algún descendiente.
- descendant-or-self : Algún descendiente o él mismo.
- following : Todos los nodos que empiezan tras el fin del actual. No incluye nodos atributo o "namespaces"
- following-sibling : Análogo al caso anterior pero con el mismo padre.

- namespace : El "namespace" del actual.
- parent : El padre del actual.
- preceding : Lo contrario de following. También excluye atributos y "namespaces".
- preceding-sibling Análogo al caso anterior pero con el mismo padre.
- self : El actual.

Todos se usan poniéndolos al principio de la cadena de selección seguidos de "::" y luego el resto de la expresión (nombre,"/","*",comment(),node(),text(),processing-instruction()...).
Abreviaturas:
.self::node()
.parent::node()
name child::name
@name attribute::name
// /descendant-or-self::node()/
Los patrones de "match" sólo pueden usar los modos abreviados y no todos.
Referirse a los padres en "match" es ilegal.
La sintaxis completa está restringida a las expresiones.

2.4.- Tipos de expresiones.
Conjuntos de nodos.
El nodo actual es un miembro de la lista de nodos actual. Ésta lista contiene el grupo de elementos que coinciden con la misma regla. Un xsl:apply-templates o xsl:for-each.
Funciones:
position() La posición del nodo actual en la lista actual. Empieza en 1.
last() El número de nodos en el conjunto actual.
count(node-set) El número de nodos en node-set.
id(string) Conjunto de nodos cuyos ID sean los dados en string (separados con espacios). Devuelve el onjunto vacío si no encuentra nada.
key(string name, Object value) Conjunto de nodos que tienen una "key" con el valor especificado. Las "keys" se ponen con xsl:key.
document(string URI, string base)
local-name (node set)
namespace-uri (node set)
name (node set)
generate-id(node set) Da un identificador único para el primer nodo de la lista. O para el actual si no lleva argumentos.
Booleans
Puede ser true o false. Cualquier cosa se puede transformar en un Boolean, siguiendo estas reglas:
- Un número es false sólo si es 0 o NaN.
- Un conjunto de nodos sólo es false si está vacío.
- Un fragmento resultante sólo es false si está vacío.
- Una cadena sólo es false si su longitud es cero.

Puede usar estos operadores: = , != , < (&lt), > , <= (<=), >= , and , or .
Nota: El < hay que sustituirlo por <
Funciones: not() , true() , false() , lang (code) (este último devuelve true si el nodo actual tiene el mismo lenguaje -especificado por el atributo xml:lang- que el argumento code).
Números
Se guardan como dobles del IEEE 754 en coma flotante de 64 bits (todos).
Todo se puede convertir automáticamente a números, con estas reglas:
- Booleans son 1 si true, 0 si false.
- Los Strings se intentan convertir en el número que representan. Si no representan un número, se convierten en NaN.
- Los conjuntos de nodos y fragmentos resultantes se convierten a String y esto a número.

Operadores: + , - , * , div , mod .
Funciones: floor() , ceiling() , round() , sum() .
Strings
Secuencia de caracteres Unicode. Las conversiones son con la función string():
- Un conjunto de nodos se convierte usando value-of sobre el primer nodo.
- Los fragmentos de árbol se convierten aplicando value-of a un supuesto elemento que los contiene a todos.
- Un número se convierte a estilo-esuropeo. Ej: "-12" , "3.1415".
- Un boolean false se convierte en "false", y un true en "true".

Funciones:
Boolean starts-with(main_string, prefix_string)
Boolean contains(containing_string, contained_string)
string substring(string, offset, length) Cadena de tamaño length desde offset, en la cadena dada.
string substring-before(string, marker-string) Cadena de la dada, desde el principio hasta la marca, sin incluirla.
substring-after(string, marker-string) Análogo. También sin incluirla.
Number string-length(string)

```
string normalize-space(string) Cadena normalizada. Sin argumentos, normaliza el nodo
actual.
string translate(string, replaced_text, replacement_text)
string concat(string1, string2, …)
string format-number(number, format-string, locale-string) Formatos en Java 1.1:
java.text.DecimalFormat.
locale-string es opcional.
Fragmentos de árbol resultante
Son fragmentos de árboles XML, pero no es XML bien formado. Las únicas operaciones sobre
ellos son: string() , boolean().

2.5- Reglas por defecto para los templates
<xsl:template match="*|/">
<xsl:apply-templates/>
</xsl:template>
Hace que se encuentren todos los elementos. Pero si se selecciona un padre y no
explicita lo de xsl:apply-templates, entonces los hijos no se encuentran.
<xsl:template match="text()|@*">
<xsl:value-of select="."/>
</xsl:template>
Copia toda la información en texto y atributos a la salida.
<xsl:template match="processing-instruction()|comment()"/>
Ignora las instrucciones de procesamiento y los comentarios.
Nota: El XSL de Internet Explorer 5.0, no tiene reglas por defecto.

2.6.- Particularidades para dar la salida-
{} .- Para poner en los atributos de salida el valor de un elemento del documento de
entrada, se utilizan llaves alrededor de dicho elemento. Se puede colocar concatenado
con más carácteres. Lo que hay dentro de las llaves puede ser cualquier XPath.
<xsl:element> .- Para colocar un elemento en la salida, se utiliza la etiqueta
<xsl:element name="{...}" >. El atributo name determina el nombre del elemento que se va
a colocar en la salida.
<xsl:attribute> .- Para colocar atributos a elementos en la salida con valores variables
dependientes de la entrada, se usa la forma
<xsl:attribute name="...">
<xsl:value-of select="..."/>...
</xsl:attribute>
El atributo de xsl:attribute, name determina el nombre del atributo de la salida,
mientras que el contenido entre las dos etiquetas <xsl:attribute> y </xsl:attribute>,
determina el valor de dicho atributo.
Todos los xsl:attribute deben ir antes de cualquier otro contenido del elemento padre.
<xsl:attribute-set> Define un grupo de atributos en el nivel superior de la hoja de
estilo.
Ej:
<xsl:attribute-set name="...">
<xsl:attribute name="...">
...
</xsl:attribute>
<xsl:attribute-set>
<xsl:use-attribute-sets> Con esto se utiliza uno o más grupos de atributos definidos con
<xsl:attribute-set>. Si hay más de uno se separan con espacios en la cadena que los
invoca. Ej:
<xsl:use-attribute-sets="... ... ...">
También se pueden usar los grupos de atributos como atributos de un xsl:element,
xsl:copy o xsl:attribute-set; en cuyo caso tan sólo se pone use-attribute-sets, sin
xsl:.
<xsl:processing-instruction> Para incluir una instrucción de procesamiento en la salida.
Su atributo name indica la instrución objetivo, y el contenido indica los argumentos de
la instrucción.
<xsl:comment> Para incluir comentarios en la salida. El contenido del comentario será el
contenido de la etiqueta <xsl:comment>.
<xsl:text> Sirve para introducir texto en la salida. Las ventajas sobre sacarlo sin esta
etiqueta son:
- Controlar el espaciado cuando éste es importante.
- Poner un atributo disable-output-escaping="yes" en la etiqueta para que el texto que
se de como salida no se coloque como secuencias de escape . Aunque nosotros tenemos que
colocarlo con ellas para que nuestro documento XSL esté bien formado.
<xsl:copy> Copia el nodo de la fuente en la salida tal cuál. Los elementos, atributos y
comentarios de los hijos no se copian automáticamente. Util para transformar XML en otro
XMl parecido. Ej, transformación identidad:
<xsl:template match="*|@*|comment()|processing-instruction()|text()">
<xsl:copy>
<xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
```

```
</xsl:copy>
</xsl:template>
```
Deja el documento tal cuál.
<xsl:copy-of> xsl:copy sólo copia el propio nodo. xsl:copy-of en cambio puede seleccionar con select.
<xsl:number> Coloca un entero en la salida. Su valor viene dado por el atributo value y el formato por el atributo format. Los dos atributos tienen valores por defecto. El valor por defecto de value tiene relación con la posición del nodo actual en el árbol, y puede ser ajustado por los atributos:
level
count
from
Otros atributos:
format Valores: i , I (romanos) , a , A (letras). Se puede fijar desde que número o letra empezar, poniendo el valor del atributo format como el valor inicial en el formato que lo vamos a querer. Ej: format="iii". Si se quieren colocar ceros a la izquierda del número, se escribe en format también. Ej: "01".
letter-value Especifica si se quiere numerar con letras (alphabetic) o con números (traditional).
grouping-separator Tiene el valor del signo a colocar entre los grupos de números que se hacen cuando el valor del número es muy grande.
grouping-size Define el número de dígitos en cada grupo de esos. Ej: En España sería, <xsl:number grouping-separator="." grouping-size="3"/>, supongo.
<xsl:sort> Se coloca como hijo de un xsl:apply-templates o un xsl:for-each, para ordenar la salida de los elementos seleccionados por ellos. Atributos:
select .- Nos dice en qué nos tenemos que fijar para ordenar alfabéticamente. Por defecto, en el orden lógico. Ej: select="ATOMIC_NUMBER".
data-type .- Si se pone "number", ordena numéricamente.
order .- Si se pone "descending" lo hace de manera descendente.
case-order .- Si se pone "upper-first" irán primero las mayúsculas, y con "lower-first" lo contrario. El valor defecto depende del lenguaje.
Modos .- Para dar salida en distintos puntos para la misma entrada, se utilizan los modos. Se pueden definir varios modos, incluyendo un atributo mode en xsl:aplly-templates, y utilizando el nombre del modo que nos interese cada vez, colocando un atributo mode en xsl:template, para sólo procesarlos cuando coincida el modo. Por defecto, por cada modo que inventamos, el formateador XSL añade una regla de aplicación:
```
<xsl:template match="*|/" mode="n">
<xsl:apply-templates mode="n"/>
</xsl:template>
```
Como todo esto por defecto, se puede pisar.
<xsl:variable> Para definir un valor referenciable. Su atributo name da el nombre de la variable, y su contenido el valor que se le asigna. Para referenciarla se utiliza el símbolo "$". Ej:
```
<xsl:variable name="copy99">
Copyright 1999 Elliotte Rusty Harold
</xsl:variable>
<BLOCK COPYRIGHT="{$copy99}">
</BLOCK >
<xsl:value-of select="$copy99"/>
```
Puede tener marcas, y referencias a otras variables, pero no puede referenciarse a sí misma directa o indirectamente. Existe jerarquía en cuanto a variables globales y locales.
Si dos variables con el mismo ámbito coinciden en nombre, se referencia la más cercana.
<xsl:template name="..."> Para definir macros. Es como xsl:variable, pero puede hacer uso de los valores del elemento actual. El atributo name es el nombre. Para referenciarlo se usa la etiqueta
<xsl:call-template name="..."/> alla donde se quiera usar.
<xsl:param> Se coloca como hijo de un xsl:template para pasar parámetros. En los elementos xsl:call-template o xsl:apply-templates, los parámetros se representan como hijos de nombre,
<xsl:with-param> .- Un atributo name da el nombre del parámetro, y el contenido da un valor por defecto en caso de que el invocante no lo de.
<xsl:space>, <xsl:strip-space>, <xsl:preserve-space> .- Son etiquetas para controlar si se conservan los espacios en blanco, y dónde lo hacen.
<xsl:if test="..."> .- Sólo se procesará si se el valor Boolean que da el atributo test es true.
<xsl:choose> .- Selleciona una de las posibles salidas dadas por sus hijos
<xsl:when test="..."> Si más de uno de ellos es cierto, sólo se elige el primero. Si ninguno es cierto, se procesa el
<xsl:otherwise>.
<xsl:import href="..."> .- Es un elemento de alto nivel y ha de ir en el elemento raíz xsl:stylesheet. Permite importar una o varias XSL. Las precedencias en los conflictos de reglas son. La presente XSL tiene prioridad sobre las otras, que tienen más prioridad cuanto más tarde sean importadas.

```
<xsl:apply-imports> .- Es como xsl:apply-templates pero sólo se aplica sobre lo
importado.
<xsl:include href="..."> .- Copia otro style-sheet o un elemento suyo en el presente
XML. A efectos, es como si estuviera copiado y pegado en el presente documento, lo cuál
quiere decir que lo que se incluye de esta forma tiene la misma precedencia que lo
realmente escrito.
Nota: Se puede unir en un mismo documento el XML y el XSL, pero no es recomendable.
<xsl:output method="..."/> .- Sirve para definir al principio del documento en que
formato queremos dar el documento de salida. Los valores de method son: xml, html, text.
Otros atributos permiten controlar mejor el documento xml de salida:
omit-xml-declaration Puede valer "no" o "yes", y escribirá o no una declaración XML al
principio del documento de salida.
version Por defecto "1.0". En el futuro será útil usarlo.
standalone Se pone el valor que tendrá el atributo standalone del documento de salida.
encoding Define la codificación de salida deseada.
doctype-system Para incluir una línea <!DOCTYPE en el documento de salida.
doctype-public Idem.
indent Por defecto "no". Si es "yes", se permite al procesador de XSL incluir sangrado
para que el documento quede mejor (lo intenta, pero no está obligado).
cdata-section-elements Se le pone una cadena de los nombres de los tipos de elementos
(separados por espacios) cuyo contenido debe ser tratado como CDATA.
media-type Especifica el tipo MIME del documento de salida.
]]>
</contenido>
</tema>

<tema>
<subÍndice id="SIxhtml">
Introduccion a XHTML.

Contenido:
<entrada href="xhtml">1.- Introducción. </entrada>
<entrada href="xhtml#dXHTML">2.- Diferencias con HTML 4. </entrada>
<entrada href="xhtml#hXHTML">3.- Hacia donde evolucionará XHTML. </entrada>
<entrada href="xhtml#aXHTML">Apéndices. </entrada>
</subÍndice>
<contenido id="xhtml">
Introduccion a XHTML.

1.- Introducción.

1.1.- Qué es XHTML.
· HTML desde sus orígenes (cuando Tim-Barners-Lee lo desarrolló en el CERN), se ha
definido en términos de SGML . Ahora, con la llegada de XML , se ha replanteado la
situación, y se ha buscado una alternativa a HTML definida en términos de XML: XHTML .
· Tal y como lo entiende la W3 : XHTML es una familia de tipos de documento y módulos
que reproducen, engloban y extienden HTML 4. La familia de tipos de documentos de XHTML
están definidos con XML, y están diseñados para trabajar con los navegadores basados en
XML.

1.2.- Qué me ofrece XHTML.
· Según la W3, los desarrolladores que se pasen a XHTML 1.0 obtendrán los beneficios
siguientes:
- los documentos XHTML están totalmente integrados en XML: serán visionados, editados y
validados con herramientas XML estándar;
- los documentos XHTML pueden funcionar tan bien (ó mejor) que los documentos HTML 4 en
los navegadores existentes, así como en los navegadores que trabajen con XHTML.
- los documentos XHTML pueden utilizar aplicaciones (applets y scripts) que operen sobre
el modelo de objetos del documento de HTML o XML (DOM).
· XHTML ofrece (en general) las siguientes ventajas:
- los desarrolladores de los documentos, y los diseñadores de navegadores están
continuamente encontrando nuevas formas de expresar sus ideas con marcado XML. En XML es
relativamente simple introducir nuevos elementos o atributos; la familia XHTML está
diseñada para acomodar estas extensiones con módulos XHTML; estos módulos permitirán la
combinación de lo que ya exista, y de las nuevas incorporaciones cuando se esté
desarrollando nuevos contenidos, y cuando se estén diseñando nuevos navegadores; en tres
palabras: XHTML es extensible.
- se están introduciendo constantemente formas alternativas de acceso a internet
(teléfonos móviles, televisores, video-consolas,...); según algunas estimaciones, en el
año 2002, el 75% de los accesos a documentos de Internet se realizará desde estos
aparatejos; la familia XHTML está diseñada pensando en navegadores de cualquier tipo (es
general).

<destino name="dXHTML">2.- Diferencias con HTML 4.</destino>
```

```
<![CDATA[
2.1.- Por qué hay diferencias entre HTML 4 y XHTML.
· La principal razón es que XHTML es una aplicación XML, y ciertas prácticas que eran
perfectamente legales en HTML 4 (basado en SGML) han cambiado.

2.2.- Repaso de algunas diferencias
- Los documentos deberán estar bien formados : esencialmente todos los elementos deben
estar encerrados entre etiquetas de principio y fin; y todos los elementos deben estar
anidados. Aunque solapar marcas es ilegal también en SGML, la mayoría de los browsers
toleran cosas del tipo:
ejemplo: <p>esto ya no será válido en <em>XHTML<p></em>,
que ya no podrán darse en XHTML.
- Nombres de elementos y atributos tendrán que estar en minúscula: esto es necesario
debido a que XML es "case-sensitive" (<li> y<LI> son etiquetas distintas).
- Para los elementos no vacíos, es obligatoria la marca de fin: en algunos elementos se
permitía omitir la marca final, si la marca que va después involucraba el cierre de la
anterior:
ejemplo: <p>Esto ya no será válido <p> en XHTML.
- Los valores de los atributos deben estar entre comillas, (aunque su contenido sea
numérico).
- No se permite la minimización de atributos : en HTML se permitía que nombres de
atributos (compact o checked) no necesitasen la especificación de su valor, bastaba con
poner el atributo para inferir el valor que tenía; esto no será válido en XHTML.
- Los elementos vacíos deberán tener etiqueta de final o deberán usar la etiqueta de
elemento vacío (la que acaba con "/>"); ya no se podrá usar <br>, deberá usarse
<br></br> ó <br/>.
- En los valores de los atributos, los navegadores quitarán los espacios del principio y
los del final; y pasarán la secuencia de espacios (y otros separadores: tabuladores,
saltos de línea, ...) a un solo espacio.
- Los elementos script y style están declarados como #PCDATA, por lo tanto "<" y "&" se
tratatarán como marcas de inicio, (para evitarlo utilizar secciones CDATA).
- Con SGML, se puede expresar en la DTD que determinados elementos no pueden estar
incluídos en otros, esto no se puede especificar en XML. Como consecuencia de esto,
restricciones que están incluidas en la definición estricta de HTML4 (como no poder
anidar elementos 'a' dentro de otros elementos 'a'), no estarán presentes en XHTML.
Estas prohibiciones se recogen explícitamente en un apéndice.
- Algunos elementos de HTML 4, (como: a, applet, form, frame, iframe, img y map), tienen
un atributo " name "; y por otro lado HTML 4 también incluye el atributo " id " para
todos los elementos; en ambos casos, este atributo se usa como identificador del
atributo. En XML sólo se usará el atributo "id".
]]>
<destino name="hXHTML">3.- Hacia donde evolucionará XHTML.</destino>

3.1.- Modularización en XHTML.
- XHTML proporciona las bases para una familia de tipos de documentos que extenderán y
englobarán a XHTML, para poder soportar una amplio rango de dispositivos y aplicaciones.
Gracias a la definición en varios módulos, podrán utilizarse para distintas plataformas:
distintos elemementos, (al tener XHTML definidos un conjunto de elementos mas pequeños;
podemos combinarlos según las necesidades que tengamos en cada caso). La división en
módulos está todavía por definir (en breve en la W3).

3.2.- Subconjuntos y extensiones.
- Modularizar nos lleva a varias ventajas:
· nos proporciona un mecanismo formal para usar subconjuntos de XHTML;
· nos proporciona un mecanismo formal para extender XTHML;
· simplifica la transformación entre documentos;
· facilita la reutilización de módulos en nuevos tipos de documento.

3.3.- Profiles.
.

<destino name="aXHTML">Apéndices.</destino>

A.1.- Las DTDs de XHTML.
.

A.2.- Prohibiciones en XHTML.
- Como ya comentamos arriba, hay elementos con restricciones a cerca de su contenido, en
concreto:
· a: no puede contener otros elementos a;
· pre: no puede contener los elemementos (img, object, big, small, sub ó sup);
· button: no puede contener los elementos (input, select, textarea, label, button, form,
fieldset - es decir, cualquier otro elemento de un form- , iframe ó isindex);
· label: no puede contener otros elementos label;
```

```
· form: no puede contener otros forms.
</contenido>
</tema>
</contenidos>
```

C.3.3 Documento de la aplicación

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="tutorialesXML">

   <ventana id= "v1">
      <nombre>ventana 1</nombre>

      <panel id= "p1.1" tam=" 140, 320">
        <contenidoPanel>
         <contenidoDefecto>/contenidos/índice</contenidoDefecto>
        </contenidoPanel>
          <destinoEnlaces panel="p1.2"/>
      </panel>

      <panel id= "p1.2" tam="320, 200">
        <contenidoPanel>
         <contenido>/contenidos/tema[1]/subÍndice</contenido>
         <contenido>/contenidos/tema[2]/subÍndice</contenido>
         <contenido>/contenidos/tema[3]/subÍndice</contenido>
         <contenido>/contenidos/tema[4]/subÍndice</contenido>
         <contenido>/contenidos/tema[5]/subÍndice</contenido>
        </contenidoPanel>
         <destinoEnlaces panel="p1.3"/>
      </panel>

      <panel id= "p1.3" tam="400, 500">
        <contenidoPanel>
         <contenido>/contenidos/tema[1]/contenido</contenido>
         <contenido>/contenidos/tema[2]/contenido</contenido>
         <contenido>/contenidos/tema[3]/contenido</contenido>
         <contenido>/contenidos/tema[4]/contenido</contenido>
         <contenido>/contenidos/tema[5]/contenido</contenido>
        </contenidoPanel>
      </panel>
   </ventana>

</aplicacion>
```

C.4 Curso IMS

C.4.1 DTD de contenidos

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT curso (índice, temas, ejercicio+, corrección+)>
<!ATTLIST curso título CDATA #REQUIRED>

<!ELEMENT índice (título, entrada+)>
<!ELEMENT título (#PCDATA)>
<!ELEMENT entrada (#PCDATA)>
<!ATTLIST entrada href IDREF #IMPLIED>

<!ELEMENT temas (tema+)>
<!ELEMENT tema (título, contenido)>
<!ATTLIST tema id ID #REQUIRED>
<!ELEMENT contenido (#PCDATA|imagen|enlace)*>
<!ELEMENT imagen EMPTY>
<!ATTLIST imagen imagen CDATA #REQUIRED>
<!ELEMENT enlace (#PCDATA)>
<!ATTLIST enlace href IDREF #REQUIRED>

<!ELEMENT ejercicio (pregunta+, corregir)>
<!ATTLIST ejercicio id ID #REQUIRED>
<!ELEMENT pregunta (texto, respuesta+)>
<!ELEMENT texto (#PCDATA)>
<!ELEMENT respuesta (#PCDATA)>
<!ELEMENT corregir (#PCDATA)>
<!ATTLIST corregir href IDREF #REQUIRED>

<!ELEMENT corrección (#PCDATA|enlace)*>
<!ATTLIST corrección id ID #REQUIRED>
```

C.4.2 Documento de contenidos

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE curso SYSTEM "curso.dtd">

<curso título="Curso de IMS">

  <índice>
    <título>IMS Global Learning Consortium</título>
    <entrada href="queEs">1. ¿ Qué es IMS Global Learning Consortium ?</entrada>
    <entrada href="proceso">2. Proceso de desarrollo de las especificaciones</entrada>
    <entrada href="asunciones">2.1. Asunciones pre-diseño de las
especificaciones</entrada>
    <entrada href="conceptos">2.2. Algunos conceptos usados por IMS</entrada>
    <entrada href="espec">2.3. Especificaciones que lo componen</entrada>
    <entrada>3. IMS Content Packaging Information Model</entrada>
    <entrada href="intro">3.1. Introducción</entrada>
    <entrada href="contentP">3.2. Modelo conceptual del IMS Content Packaging</entrada>
    <entrada href="metadata">4. IMS Learning Resource Meta-data Information
Model</entrada>
    <entrada href="enterprise">5. IMS Enterprise Information Model</entrada>
    <entrada href="qYt">6. IMS Question &amp;amp; Test Interoperability Information Model
Specification</entrada>
    <entrada href="learnerI">7. Learner Information Packaging</entrada>
    <entrada href="bibliografia">8. Bibliografía</entrada>
  </índice>

  <temas>
  <tema id="queEs">
    <título>¿Qué es IMS Global Learning Consortium?</título>
    <contenido>
'IMS' significa 'Instructional Management Systems' que se podría traducir por algo así
como 'Sistemas de gestión de la enseñanza', y en un principio éste era su cometido, pero
en nuestros nuestros días IMS se está centrando en estándares para servidores de
conocimiento, para contenidos dedicados al aprendizaje, y en la introducción de estos
```

```
entándares en la empresa. Es por todo esto que IMS es solamente un nombre, y no hay que
fijarse en lo que significan sus siglas.
IMS es también un intento por conseguir una especificación para el desarrollo del
potencial de Internet como entorno de formación, permitiendo que contenidos y entornos
de aprendizaje de múltiples autores puedan trabajar juntos.
<imagen imagen="c:\plumbingXJ\GAP\documentos\imsLogo.jpeg"/>
Logo IMS Global Learning Consortium
En su dirección web http://www.imsproject.org podrás encontrar todo tipo de información
sobre sus actividades.
IMS reúne un conjunto de organizaciones académicas, comerciales y gubernamentales que
trabajan en construir la arquitectura de Internet para el aprendizaje, entre las que
están importantísimas empresas del mundo de la informática: Microsoft, Sun Microsystems,
Cisco, Apple, Oracle, IBM ....
No cabe duda de la importancia del trabajo que este grupo viene desarrollando ya que la
adopción de sus especificaciones como un estándar internacional supondrá el impulso
definitivo para los Entornos Virtuales de Enseñanza: el Aprendizaje a través de
Internet.
    </contenido>
  </tema>

  <tema id="proceso">
    <título>Proceso de desarrollo de las especificaciones</título>
    <contenido>
IMS Project enfatiza, en practicamente todos sus documentos oficiales, que su
especificación sólo define formatos y protocolos para intercambios de información que
ocurren dentro y con otros entornos IMS, pero no pretende prescribir a los
desarrolladores como deben hacer sus aplicaciones educativas ni pretende crear una
metodología instruccional.
Para el desarrollo de estas especificaciones, se asumió como punto de partida unos
requerimientos generados por los usuarios (profesores, estudiantes, proveedores,
coordinadores ) y desarrolladores de recursos de aprendizaje por Internet. Y es a partir
de ahí desde dónde se inicia la elaboración de las especificaciones.
Posteriormente estas especificaciones sufrirán adaptaciones producidas por la propia
implementación de los cursos, la cual producirá nuevos requerimientos que influirán de
nuevo en dichas especificaciones, de manera directa ( IMS se da cuenta de sus
limitaciones ) o indirecta ( son los usuarios y desarrolladores los que informan a IMS
), cerrándose así el ciclo evolutivo.
<imagen imagen="c:\plumbingXJ\GAP\documentos\evolucionIMS.gif"/>
    </contenido>
  </tema>

  <tema id="asunciones">
    <título>Asunciones pre-diseño de las especificaciones</título>
    <contenido>
1.-Los contenidos necesitan ser modularizados, el sistema debe ser escalable: Este
principio es una consecuencia lógica de entender el aprendizaje como un conjunto de
intercambios entre partes. Supone que los programas de aprendizaje deben ser pensados
como una colección de unidades de aprendizaje independientes permitiendo diferentes
combinaciones para diferentes contextos y diferentes alumnos. Esta habilidad para
modularizar los contenidos facilitará un incremento en la producción y calidad de los
materiales.
2.-Deseo de interoperatividad: Garantiza que una herramienta construida para trabajar en
un ambiente de aprendizaje, lo hará también en otros.
3.-Deseo de Personalizar y Extender: La especificación de IMS estará pensada para que
los diseñadores la puedan adaptar a los requerimientos específicos de sus usuarios. Las
especificaciones ayudan a definir una base común, desde la cual diferentes comunidades
de aprendizaje pueden desarrollar sus propios requerimientos y especificaciones.
Respecto a la extensibilidad la especificación debe esta abierta a la incorporación de
nuevos estamentos, etiquetas de metadatos... en la medida en que el desarrollo y las
necesidades de los usuarios así lo determine.
    </contenido>
  </tema>

  <tema id="conceptos">
    <título>Algunos conceptos usados por IMS</título>
    <contenido>
Contenido: Puede ser un módulo educativo o un conjunto de varios módulos con una mismo
objetivo. Por otra parte no hay una cantidad mínima de información, es decir, el
'contenido' podría ser una única palabra.
Contenedores: Encapsulan una cantidad dada de contenido. Pueden ser un elemento primario
(texto, vídeo, gráficos, herramientas) y/o otros contenedores. Para facilitar el
almacenamiento, y búsqueda, cada contenedor tendrá un juego de campos de metadatos.
Metadatos: Son datos acerca de los datos. Es información descriptiva acerca de recursos
de aprendizaje.
```

```
    </contenido>
  </tema>

  <tema id="espec">
    <título>Especificaciones que lo componen</título>
    <contenido>
Content Packaging Specification v1.1: Proporciona soporte al intercambio de contenidos
de carácter educativo entre diferentes autores, publicadores y otros desarrolladores.
Meta-Data Specification v1.1: Describe los nombres, definiciones, organización y
restricciones de los elementos Meta-datos de IMS.
Enterprise Specification v1.01: Da soporte a la interacción entre los sistemas de
gestión de información y los sistemas empresariales tales como gestión de recursos
humanos, administración de estudiantes y sistemas de gestión de bibliotecas.
Question &amp;amp; Test Specification v1.1: Proporciona soporte al intercambio de
cuestionarios, tests, exámenes... entre diferentes autores, publicadores y otros
desarrolladores de contenido.
Learner Information Packaging v1: Es el más reciente de los estándares (su única versión
es de Marzo de 2001 )

--------------------------------------------------------------------------------
<enlace href="ejercicio1">Ejercicio 1: Repaso de todos los aspectos vistos.</enlace>
    </contenido>
  </tema>

  <tema id="intro">
    <título>Introducción</título>
    <contenido>
"El IMS Content Packaging Information Model describe estructuras de datos que se
utilizan para proporcionar interoperatividad del contenido basado en Internet con
herramientas para la creación de dicho contenido, sistemas de gestión de información y
entornos en tiempo de ejecución" ( IMS Content Packaging Information v1.0, Introduction
)
Su objetivo es definir y estandarizar un conjunto de estructuras que permitan el
intercambio de contenido. La versión 1.0 de esta especificación está orientada a la
exportación, importación, agregación y eliminación de paquetes de contenido entre
diferentes sistemas de gestión de información. Pero es probable que futuras versiones de
éste estándar también hagan referencia a la comunicación en tiempo real entre los
diferentes sistemas.
    </contenido>
  </tema>

  <tema id="contentP">
    <título>Modelo conceptual del IMS Content Packaging</título>
    <contenido>
El modelo conceptual es el siguiente:
<imagen imagen="c:\plumbingXJ\GAP\documentos\package.jpeg"/>
Diagrama correspondiente a un 'paquete'

Está formado por dos partes: Un fichero XML ( llamado 'manifiesto IMS' ) que describe la
organización del contenido y los recursos del paquete, y una segunda parte formada por
los ficheros físicos descritos por el XML. Al fichero compuesto por estas dos partes se
le denomina 'fichero de intercambio de paquete', o simplemente 'Paquete'.
El manifiesto, cuyo nombre será 'imsmanifest.xml' por convenio, está compuesto por:

- Sección de meta-datos: Un elemento de XML que describe el manifiesto como una unidad.
- Sección de organizaciones : Uno o más elementos del XML que describen organizaciones
del contenido dentro de un manifiesto.
- Sección de recursos: Un elemento del XML que hace referencia a todos los recursos
necesitados por el manifiesto, incluyendo meta-datos sobre esos recursos, y referencias
a ficheros externos.
- Submanifiestos: Permiten el anidamiento de manifiestos.

Los ficheros físicos podrán ser de cualquier tipo.


Elementos de un manfiesto:
<imagen imagen="c:\plumbingXJ\GAP\documentos\maniElem.gif"/>
Elementos de un manifiesto
Aquí se pueden observar todos los elementos que componen un manifiesto, y la
interrelación que se da entre ellos.
    </contenido>
  </tema>

  <tema id="metadata">
```

```
    <título>IMS Learning Resource Meta-data Information Model</título>
    <contenido>
Los desarrolladores y diseñadores de materiales destinados al aprendizaje disponen de
una gran variedad de herramientas software para crear esos recursos de aprendizaje.
Desafortunadamente, esta gran varidad de herramientas de distintos vendedores, produce
materiales de enseñanza que no comparten los mismos mecanismos de búsqueda de recursos.
Se podrían usar etiquetas para indexar los recursos de aprendizaje y hacer, por tanto,
más fácil su búsqueda y uso. Tales etiquetas son 'datos sobre los datos', es decir, meta
datos ( meta-data ). Un ejemplo de meta datos es, por ejemplo, la etiqueta de una lata
de comida, la cual describe los ingredientes, peso, coste ... de la lata.
Otro ejemplo es la tarjeta de un libro en una biblioteca, la cual describe el libro, su
autor, tema, localización dentro de la librería...
Por tanto, una especificación que hace uso de meta datos, hace el proceso de búsqueda y
uso de recursos mucho más eficiente, mediante el uso de una estructura que describa los
recursos de aprendizaje.
    </contenido>
  </tema>

  <tema id="enterprise">
    <título>IMS Enterprise Information Model</título>
    <contenido>
El IMS Enterprise Information Model describe estructuras de datos que se utilizan para
proporcionar la interoperatividad de sistemas de gestión de recursos de aprendizaje
encaminados a Internet con otros sistemas de la empresa usados para dar soporte a las
operaciones de la organización.
El objetivo del IMS Enterprise Information Model es definir un conjunto de estructuras
que se puedan usar para intercambiar datos entre diferentes sistemas. Estas estructuras
proporcionanarán la base para la transmisión de información entre software desarrollado
por distintos proveedores.
Los principales tipos de aplicaciones empresariales soportadas por este estándar son la
administración de los modelos de enseñanza, la administración de los estudiantes, el
mantenimiento de la 'biblioteca' de información, y los sistemas de recursos humanos.
Cabe destacar que el objetivo del IMS Enterprise Information Model está centrado en la
definición de la interoperatividad entre sistemas que estén dentro de la misma empresa y
no en resolver los problemas de integridad de datos, seguridad de información y otros
problemas relativos al intercambio de datos.
    </contenido>
  </tema>

  <tema id="qYt">
    <título>IMS    Question    &amp;amp;    Test    Interoperability    Information    Model
Specification</título>
    <contenido>
El IMS Question &amp;amp; Test Interoperability Information Model Specification describe
estructuras de datos que se utilizan para proporcionar interoperatividad entre sistemas
de test y preguntas, particularmente aquellos que están basados en Internet.

Existen tres partes básicas:

- Assessment: La unidad básica o test.
- Sección: Un contenedor de grupos de secciones e items que soportan un objetivo común.
- Item: El bloque más básico compuesto de pregunta y respuesta.

La principal ayuda que proporciona esta especificación es permitir a los usuarios la
importación y exportación de sus preguntas ( llamadas 'items' y agrupadas en secciones )
y tests ( llamados 'assesmentes' y que contienen secciones ).
Todo esto requiere un especificación concisa y sin ambigüedades que cubra el más amplio
rango posible de tipos de preguntas y tests. Esta especificación está costruida de tal
forma que es capaz de soportar tanto tests complejos como simples, y permitir
extensiones del usuario que no comprometan la integridad del sistema.
Por ejemplo, lo siguiente es la taxonomia para los posibles tipos de respuesta
<imagen imagen="c:\plumbingXJ\GAP\documentos\QTItipoRespuesta.gif"/>
Taxonomía para una 'respuesta'

- Basic : Sóla hay un tipo de respuesta
- Composite: Hay varios tipos de respuesta
- Time Dependent : Es importante el tiempo que se tarde en responder.
- Time Independent : No es importante el tiempo que se tarde en responder.
    </contenido>
  </tema>

  <tema id="learnerI">
    <título>Learner Information Packaging</título>
    <contenido>
```

```
Learner Information' es una colección de información sobre un 'Learner' ( un estudiante
o grupo de estudiantes ) o un productor de contenido para el aprendizaje ( creadores,
proveedores o vendedores ).
La especificación del 'IMS Learner Information Package' se dirige a la interoperatividad
entre los sistemas de información del estudiante basados en Internet, con otros sistemas
que también soporten el medio de aprendizaje en Internet.
La especificación intenta definir un conjunto de paquetes que se puedan usar para
importar datos y extraerlos de un servidor de información compatible con IMS.
Es responsabilidad del servidor de información del 'Learner' el permitir al propietario
de la información definir que información quiere compartir con otros sistemas;
información relativa a fines, intereses, calificaciones, certificaciones, licencias,
relaciones, claves de seguridad ...
    </contenido>
  </tema>

  <tema id="bibliografia">
    <título>Bibliografia</título>
    <contenido>
1.-IMS Global Learning Consortium : http://www.imsproject.org
2.-Edutec. Revista Electrónica de Tecnología Educativa. Núm. 13.
    </contenido>
  </tema>
  </temas>

  <ejercicio id="ejercicio1">
    <pregunta>
      <texto>1. ¿Qué es IMS Global Learning Consortium?</texto>
      <respuesta>Es un intento por conseguir una especificación para el desarrollo del
potencial de Internet como entorno de formación.</respuesta>
      <respuesta>Es un sistema de gestión de la enseñanza.</respuesta>
      <respuesta>Es un consorcio de aprendizaje global sobre un estándar mundial sobre
Internet.</respuesta>
    </pregunta>
    <pregunta>
      <texto>2. El proceso de desarrollo de las especificaciones de IMS...</texto>
      <respuesta>Es evolutivo, y a él sólo contribuyen los miembros de IMS.</respuesta>
      <respuesta>Es evolutivo, e influyen tanto miembros de IMS como desarrolladores de
Software educativo y usuarios.</respuesta>
      <respuesta>Una vez alcanzado el estándar, quedará cerrado.</respuesta>
       <respuesta>Es evolutivo, e influyen tanto miembros de IMS como desarrolladores de
Software educativo.</respuesta>
    </pregunta>
    <pregunta>
      <texto>3. Los Meta-Datos son:</texto>
      <respuesta>Etiquetas que se añaden a la información.</respuesta>
      <respuesta>Son casi datos.</respuesta>
      <respuesta>Son datos sobre datos</respuesta>
       <respuesta>No están cotemplados en IMS</respuesta>
      <respuesta>Son datos poco importantes para IMS y por ese se les llama 'Meta-
Datos'</respuesta>
    </pregunta>
    <corregir href="cEjercicio1">Corregir</corregir>
  </ejercicio>

  <corrección id="cEjercicio1">Contenido estático que simula la correción dinámica.
<enlace href="espec">Volver</enlace>
  </corrección>

</curso>
```

C.4.3 Documento de la aplicación

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="cursoIMS">

   <ventana id= "v1">
      <nombre>Ventana1</nombre>

      <panel id= "p1.1" tam="350, 300">
```

```
        <contenidoPanel>
         <contenidoDefecto>/curso/índice</contenidoDefecto>
        </contenidoPanel>
         <destinoEnlaces panel="p1.2"/>
      </panel>

      <panel id= "p1.2" tam="650, 700">
         <contenidoPanel>
         <contenidoDefecto>/curso/temas/tema[@id="intro"]</contenidoDefecto>
         <contenido>/curso/ejercicio</contenido>
         <contenido>/curso/corrección</contenido>
         <contenidoGrupo>/curso/temas</contenidoGrupo>
      </contenidoPanel>
         <destinoEnlaces panel="p1.2"/>
      </panel>
   </ventana>

</aplicacion>
```

C.5 Amazon

C.5.1 DTD de contenidos

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT contenidos (formulario, resultado, libros, carritos)>

<!ELEMENT formulario (texto, botón)>
<!ELEMENT texto (#PCDATA)>
<!ELEMENT botón (#PCDATA)>
<!ATTLIST botón href IDREF #REQUIRED>

<!ELEMENT resultado (#PCDATA|enlace)*>
<!ATTLIST resultado id ID #REQUIRED>
<!ELEMENT enlace (#PCDATA)>
<!ATTLIST enlace href IDREF #REQUIRED>

<!ELEMENT libros (libro+)>
<!ELEMENT libro (#PCDATA|enlace)*>
<!ATTLIST libro id ID #REQUIRED>

<!ELEMENT carritos (carrito+)>
<!ELEMENT carrito (#PCDATA)>
<!ATTLIST carrito id ID #REQUIRED>
```

C.5.2 Documento de contenidos

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE contenidos SYSTEM "cAmazon.dtd">

<contenidos>

<formulario>
<texto>Esto es el formulario de búsqueda</texto>
<botón href="res">Buscar</botón>
</formulario>

<resultado id="res">Este es el resultado con enlaces a
<enlace href="libro1">Libro 1</enlace>
<enlace href="libro2">Libro 2</enlace>
..............
<enlace href="libron">Libro n</enlace>
</resultado>

<libros>
<libro id="libro1">Este es el libro 1 que enlaza con el <enlace
href="c1">carrito</enlace></libro>
<libro id="libro2">Este es el libro 2 que enlaza con el <enlace
href="c2">carrito</enlace></libro>
<libro id="libron">Este es el libro n que enlaza con el <enlace
href="cn">carrito</enlace></libro>
</libros>

<carritos>
<carrito id="c1">Ahora se añadiría el libro 1 al carrito</carrito>
<carrito id="c2">Ahora se añadiría el libro 2 al carrito</carrito>
<carrito id="cn">Ahora se añadiría el libro n al carrito</carrito>
</carritos>
</contenidos>
```

C.5.3 Documento de la aplicación

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE aplicacion SYSTEM "aplicacion.dtd">

<aplicacion id="amazon">
```

```
   <ventana id= "v1">
      <nombre>ventana 1</nombre>

      <panel id= "p1.1">
        <contenidoPanel>
         <contenidoDefecto>/contenidos/formulario</contenidoDefecto>
        </contenidoPanel>
      <destinoEnlaces panel="p2.1"/>

      </panel>
   </ventana>


   <ventana id= "v2">
      <nombre>ventana 2</nombre>

      <panel id= "p2.1">
        <contenidoPanel>
         <contenido>/contenidos/resultado</contenido>
        </contenidoPanel>
         <destinoEnlaces panel="p3.1"/>
      </panel>
   </ventana>


   <ventana id= "v3">
      <nombre>ventana 3</nombre>

      <panel id= "p3.1">
        <contenidoPanel>
         <contenidoGrupo>/contenidos/libros</contenidoGrupo>
        </contenidoPanel>
        <destinoEnlaces panel="p4.1"/>
      </panel>
    </ventana>

   <ventana id= "v4">
      <nombre>ventana 4</nombre>

      <panel id= "p4.1">
        <contenidoPanel>
         <contenidoGrupo>/contenidos/carritos</contenidoGrupo>
        </contenidoPanel>
      </panel>
    </ventana>

</aplicacion>
```

Apéndice D. Miscelánea

En este apéndice incluimos información de naturaleza heterogénea que es referenciada desde diversos capítulos de esta tesis.

D.1 Representación Dexter del ejemplo

```
<hipertexto>
        <componente>
                <tipo> texto </tipo>
                <UID> idBiografía </UID>
                <datos> ... </datos>
                <ancla>
                   <id> 0 </id>
                   <localización> ... </localización>
                </ancla>
                <ancla>
                   <id> 1 </id>
                   <localización> ... </localización>
                </ancla>
                <ancla>
                   <id> 2 </id>
                   <localización> ... </localización>
                </ancla>
                <ancla>
                   <id> 3 </id>
                   <localización> ... </localización>
                </ancla>
                <ancla>
                   <id> 4 </id>
                   <localización> ... </localización>
                </ancla>
                <ancla>
                   <id> 5 </id>
                   <localización> ... </localización>
                </ancla>
                <ancla>
                   <id> 6 </id>
                   <localización> ... </localización>
                </ancla>
                <presentación> ventana 1 / panel 1 </presentación>
        </componente>
        <componente>
                <tipo> foto </tipo>
                <UID> idEspaña </UID>
                <datos> ... </datos>
                <ancla>
                   <id> 0 </id>
                </ancla>
                <ancla>
                   <id> 1 </id>
                   <localización> ... </localización>
                </ancla>
                <ancla>
                   <id> 2 </id>
                   <localización> ... </localización>
                </ancla>
                <presentación> ventana 2 / panel 1 </presentación>
        </componente>
        <componente>
                <tipo> texto </tipo>
                <UID> idMadrid </UID>
                <datos> ... </datos>
                <ancla>
                   <id> 0 </id>
                   <localización> ... </localización>
                </ancla>
                <ancla>
                   <id> 1 </id>
                   <localización> ... </localización>
```

```
        </ancla>
        <presentación> ventana 2 / panel 2 </presentación>
</componente>
<componente>
        <tipo> texto </tipo>
        <UID> idLepe </UID>
        <datos> ... </datos>
        <ancla>
           <id> 0 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 1 </id>
           <localización> ... </localización>
        </ancla>
        <presentación> ventana 2 / panel 2 </presentación>
</componente>
<componente>
        <tipo> texto </tipo>
        <UID> idEstudios </UID>
        <datos> ... </datos>
        <ancla>
           <id> 0 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 1 </id>
           <localización> ... </localización>
        </ancla>
        <presentación> ventana 2 / panel 1 </presentación>
</componente>
<componente>
        <tipo> texto </tipo>
        <UID> idDocencia </UID>
        <datos> ... </datos>
        <ancla>
           <id> 0 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 1 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 2 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 3 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 4 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 5 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 6 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 7 </id>
           <localización> ... </localización>
        </ancla>

        <presentación> ventana 2 / panel 1 </presentación>
</componente>
<componente>
        <tipo> texto </tipo>
        <UID> idIsg1 </UID>
        <datos> ... </datos>
```

```
        <ancla>
           <id> 0 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 1 </id>
           <localización> ... </localización>
        </ancla>
        <presentación> ventana 2 / panel 2 </presentación>
</componente>
<componente>
        <tipo> texto </tipo>
        <UID> idIsg2 </UID>
        <datos> ... </datos>
        <ancla>
           <id> 0 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 1 </id>
           <localización> ... </localización>
        </ancla>
        <presentación> ventana 2 / panel 2 </presentación>
</componente>
<componente>
        <tipo> texto </tipo>
        <UID> idIsg91 </UID>
        <datos> ... </datos>
        <ancla>
           <id> 0 </id>
           <localización> ... </localización>
        </ancla>
        <ancla>
           <id> 1 </id>
           <localización> ... </localización>
        </ancla>
        <presentación> pantalla 2 / ventana 2 </presentación>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e1 </UID>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idEspaña </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e2 </UID>
        <especificador>
                <componente_UID> idEspaña </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e3 </UID>
        <especificador>
                <componente_UID> idEspaña </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
```

```
        <especificador>
                <componente_UID> idMadrid </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e4 </UID>
        <especificador>
                <componente_UID> idMadrid </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idEspaña </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e5 </UID>
        <especificador>
                <componente_UID> idEspaña </componente_UID>
                <ancla_id> 2 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idLepe </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e6 </UID>
        <especificador>
                <componente_UID> idLepe </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idEspaña </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e7 </UID>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 2 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idEstudios </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e8 </UID>
        <especificador>
                <componente_UID> idEstudios </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
```

```
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e9 </UID>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 3 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e10 </UID>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e11 </UID>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 4 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 2 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e12 </UID>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 5 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 3 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e13 </UID>
        <especificador>
                <componente_UID> idBiografía </componente_UID>
                <ancla_id> 6 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 4 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
```

```
        <UID> e14 </UID>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 5 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idIsg1 </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e15 </UID>
        <especificador>
                <componente_UID> idIsg1 </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e16 </UID>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 6 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idIsg2 </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e17 </UID>
        <especificador>
                <componente_UID> idIsg2 </componente_UID>
                <ancla_id> 1 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e18 </UID>
        <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 7 </ancla_id>
                <direccion> DE </dirección>
        </especificador>
        <especificador>
                <componente_UID> idIsg91 </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
        </especificador>
</componente>
<componente>
        <tipo> enlace <tipo>
        <UID> e19 </UID>
        <especificador>
                <componente_UID> idIsg91 </componente_UID>
                <ancla_id> 1 </ancla_id>
```

```
                <direccion> DE </dirección>
            </especificador>
            <especificador>
                <componente_UID> idDocencia </componente_UID>
                <ancla_id> 0 </ancla_id>
                <direccion> A </direccion>
            </especificador>
        </componente>
</hipertexto>
```

D.2 Script Production

Este documento es uno de los primeros guiones para cubrir la especificación de requisitos de la herramienta Galatea en 1996. Nótese el interés por capturar el esquema navegacional y su relación con el grafo de contenidos. Por supuesto, en aquella época, estos conceptos ni siquiera existían en la mente del autor de esta tesis. Es más, probablemente dicho autor estaría centrado en acabar su carrera sin preocuparse por cuestiones sobre especificación de requisitos de aplicaciones hipermedia.

Estructura de los guiones del curso

El alumno puede elegir el idioma en que va a recibir las instrucciones, preguntas y explicaciones.
El alumno puede elegir también el idioma de las herramientas (diccionario, gramática, etc.)
El alumno puede elegir realizar los trabajos en torno a un texto de tres formas diferentes:

1.- Seguir la secuencia de ejercicios, agrupados en etapas, que constituye una lección.

Las etapas que componen una lección son:

1.- introducción: presentación del texto y selección del modo de trabajo
2.- sentido general del texto
3.- organización del texto
3.1 personajes: relaciones y papeles
3.2 tiempo: scripts y causalidad
3.3 espacio: mapas
4.- la oración gramatical
5.- giros y modismos

2.- Entrar directamente en una de las etapas de la lección.

3.- Utilizar libremente las herramientas que facilitan la comprensión del texto: sonorización del texto, diccionario, gramática y estructuras semánticas.

Elementos necesarios para describir completamente cada una de las etapas que componen una lección:

Se define una ventana principal y un conjunto (que puede ser vacío) de ventanas auxiliares.
Las ventanas principales se corresponden en principio con las etapas y sub-etapas enumeradas en el epígrafe anterior.

Definir la ventana principal de la interfaz

Atributos de la ventana
tamaño, situación, texto, gráficos, etc.
barra de menú
botones
ventanas auxiliares
relación entre las ventanas

Contenido de la ventana
información que se presenta en la ventana
servicios que facilita la ventana (ejercicios)

Relaciones entre ventanas principales
¿Cómo se llega a esta ventana principal?
¿A que sitios se puede ir desde esta ventana principal? (acciones que implican cambio de pantalla)

Definir ventanas auxiliares
atributos de la ventana
barra de menú
botones
contenido de la ventana
relación con otras ventanas

Definir ejercicios

Descripción de cada ejercicio
Material sobre el que se realiza el ejercicio
mensajes de ayuda e instrucciones
preguntas y re-preguntas (número de oportunidades)
herramientas disponibles
evaluación de las respuestas y explicaciones
forma de finalizar del ejercicio
descripción de la ventana principal y ventanas auxiliares asociadas al ejercicio.

Descripción de las herramientas

formas de acceso a la herramienta
estructura de la información que contiene
descripción detallada de todas las funciones que realiza
ventana principal y ventanas auxiliares asociadas a la herramienta

D.3 Agradecimientos

Dicen que el saber no ocupa lugar, pero la pila de material que he utilizado para elaborar esta memoria tiene un peso cercano a los veintiocho kilos. ¡Menos mal que no ocupaba lugar! Bromas aparte, este apartado está reservado para reconocer los esfuerzos de todas aquellas personas, entes y entidades que han contribuido a la realización de esta memoria.

El apartado de agradecimientos es un apartado al que hay que prestar una especial atención. Un pequeño número de agradecimientos corre el peligro de dejarse fuera personas que deberían aparecer. Un gran número de agradecimientos puede desvirtuar el verdadero esfuerzo de otras personas. La perfección, como muchas veces, parece encontrarse en el término medio. Como yo no soy perfecto (aunque casi, eso sí) opto por incluir un gran número de agradecimientos, ya que todos los esfuerzos han contribuido en mayor o menor medida a la (¿correcta?) realización de esta memoria.

Empiezo dando gracias a Dios, por esta vida tan *maja* (plena sonaba demasiado solemne) que me esta permitiendo vivir (como diría la película, *My life so far*. O "el pajarito feliz", que dirían los más escépticos). Esperemos que el resto siga en la misma línea. Ya que nos encontramos en el apartado de deidades, parece razonable dar mi más profundo agradecimiento a Alfredo Fernández-Valmayor, y a Baltasar Fernández por su guía y apoyo durante estos largos años (a mi se me han pasado volando). Hubo veces que pareció que estaba "más fuera que dentro", pero al final, parece que "me quede dentro".

Ya en un plano más material quiero agradecer a José Luis Sierra su amistad durante este tiempo. Curiosamente, José Luis entró en el grupo de investigación como "mi más directo competidor" (o así lo veía yo), y acabó siendo uno mis más firmes apoyos. A la historia pasarán nuestras charlas en el pasillo, donde, por cierto, de vez en cuando (pero muy de vez en cuando), también hablábamos de ciencia.

Pero no solo conté con el apoyo de Alfredo, Balta, y José Luis durante estos años. También dispuse del soporte y cariño (a lo mejor me excedo un poco, pero la ocasión lo requiere) de todos los miembros de mi grupo de investigación: Juan Cigarrán, Belén Díaz, Carlos Fernández (en la diáspora), Carmen Fernández, Ana Fernández-Pampillón, Pablo Gervás, Hector Gómez, Mercedes Gómez, Pedro A. González, Luis Hernández, Juan Pavón, Antonio Sarasa, Pilar Sancho, Eva Ullán (aunque Eva no es del grupo, creo yo que está *asimilada*), y como no, Antonio Vaquero. Bueno, este párrafo sería una "cosa media" si no citase a Manuel de Buenaga, y a José María Gómez, que los primeros tiempos también cuentan.

¿Y de Jorge J. Gómez me he olvidado? No, pero pienso que merecía una mención aparte, por las "ilustres conversaciones" que en alguna ocasión hemos mantenido José Luis, Jorge, y yo mismo.

Sería injusto si no mencionase también al Departamento de Sistemas Informáticos y Programación de la Universidad Complutense de Madrid por el entorno de trabajo que me ha proporcionado durante estos años. En base a mi experiencia durante este tiempo, sin duda alguna, este es uno de los mejores departamentos de toda España.

España. Precisamente también quiero dar gracias a esta gran nación, y a sus instituciones, que han contribuido en mi desarrollo personal y profesional (en este apartado debería sonar el Himno Nacional). Ministerio de Educación y Ciencia (ahora son dos, pero bueno), Comunidad de Madrid, Ministerio de Defensa, y demás organismos que han sido decisivos a lo largo de mi formación, son destinatarios de mis más sinceros agradecimientos.

Por supuesto, también quiero mencionar aquí a todos los colegas con los que he compartido más de un congreso, charla o conferencia. En particular mando un saludo a mis amigos del Segundo y Tercer SIIE, y como no, a los miembros de EDUCAM. En todos los casos escucharon mis ideas con interés, enriqueciendo mi trabajo con sus comentarios y desarrollos personales.

Aunque he empezado dando gracias a Dios y a mis directores de tesis, no quiero olvidarme de todas aquellas personas que me han permitido encontrarme en la situación de realizar los estudios de tercer ciclo. Al menos, desde mi punto de vista, esta cuestión no es ni mucho menos trivial, y como tal quiero que conste mi agradecimiento público.

Esto se acaba, porque me estoy quedando sin nada ni nadie a quien dar las gracias. ¿O no? ¡Pues no! Quiero agradecer a mis amigos, amigas, amigas++, y amigas# por todo el cariño y comprensión mostrados durante estos años, en los que casi siempre antepuse mi carrera a mi vida personal (¡qué bonito y sacrificado me ha quedado!). Gracias por seguir ahí, aunque yo muchas veces no estuviese.

And last, but not least (aunque me hubiese gustado restringirme al español, creo yo que esta expresión inglesa es la más adecuada) quiero mostrar mi agradecimiento a mi Familia (la pongo con mayúsculas porque solo tengo una). Muchas gracias de corazón. A vosotros os hago extensivos todos los agradecimientos anteriormente mencionados. Gracias por traerme al mundo, cuidarme, y permitirme dedicarme durante todos estos años (veintinueve, si no me fallan las cuentas) a lo que más me gustaba. Gracias por vuestro apoyo y comprensión en todos los momentos (los buenos, y los malos). Gracias por aguantarme, que no es nada fácil (soy yo, con lo mucho que me quiero, y a veces ni me soporto). Gracias por vuestro cariño. Gracias por todo.