ESPECIFICACION DE TIEMPO PARA EL DISEÑO DE APLICACIONES WEB E HYPERMEDIA

Alumno
A.C. Germán Terrazas Angulo

Director de Tesis
PhD. Gustavo H. Rossi

PRESENTADO EN CUMPLIMIENTO DE LOS
REQUERIMIENTOS PARA EL GRADO DE
LICENCIATURA EN INFORMÁTICA
DE LA
UNIVERSIDAD NACIONAL DE LA PLATA
LA PLATA, ARGENTINA
MARZO 2003

A Lili, por su incondicional compañia durante los años de mi carrera.

Contenido

Índice de Figuras

Capítulo 1

Introducción

El campo de acción de las aplicaciones hypermedia, en especial el de las aplicaciones web, se encuentra en constante evolución. Muchas son las razones y entre las más importantes se destacan el avance de las tecnologías y su incorporación a innumerables disciplinas. Los ejemplos sobran y entre los más comunes se encuentran los sitios web, los sistemas de gestión administrativa y el comercio electrónico. A medida que ésto sucede, las características de los requerimientos para realizar su diseño e implementación también evolucionan. En otras palabras, cada vez se necesita especificar una mayor cantidad de aspectos del dominio de la aplicación y entre éstos se encuentran aquellos que dependen del tiempo.

Nosotros entendemos que durante el proceso de desarrollo de aplicaciones hypermedia, en particular de aplicaciones web, es imprescindible realizar la especificación de cualquier aspecto temporal porque son tan importantes como cualquier otro aspecto de la aplicación. Por tal motivo, el objetivo de este trabajo consiste en realizar la definición de mecanismos útiles para poder especificar aspectos temporales en la etapa de diseño.

1.1 Introducción a Hypermedia

Para la mayoría de los usuarios la experiencia más cercana al uso de hypermedia es la web. Los sitios web son un tipo de aplicaciones hypermedia que tienen como plataforma de soporte el World Wide Web y la gran mayoría no alcanza a mostrar la totalidad de las características de ese tipo de aplicaciones.

Dentro del marco teórico existen varias interpretaciones de hypermedia. La más simple es aquella que lo define como un tipo de aplicación que permite al usuario navegar a través de todo el espacio de información del sistema de manera no lineal. Una forma de entender éste concepto es usando un ejemplo que describa lo contrario como por ejemplo la organización de la información contenida en un libro. En un libro la organización de la información es lineal por la sencilla razón de que no hay manera de recorrerla por completo sino es a través de la lectura secuencial de sus capítulos. En otras palabras, en un libro existe un solo camino posible para recorrer el espacio de información mientras que en las aplicaciones hypermedia existe más de uno.
Otra interpretación diferente define hypermedia como un tipo de aplicación que integra hypertext con multimedia. En éste se define a las aplicaciones multimedia como aquellas donde la presentación de la información se realiza usando diferentes medios como por ejemplo sonido, imágenes, texto o video. Ejemplos de este tipo de aplicaciones son las usadas para el aprendizaje interactivo o las enciclopedias. Por otro lado, las aplicaciones hypertext se definen como aplicaciones que se basan en la presentación de la información a base de documentos de texto relacionados entre sí de manera no lineal (hyperdocuments). Una manera abstracta de ver estas aplicaciones es mediante un conjunto de nodos y un conjunto de enlaces. Así, cada nodo representa un documento y cada enlace una relación entre pares de nodos.

Las aplicaciones hypermedia reúnen tres características importantes: interactúan con el usuario, combinan distintos tipos de medios y poseen una estructura no lineal. Desde un punto de vista arquitectónico, en una aplicación hypermedia pueden convivir varias relaciones entre las entidades de información. Por ejemplo, relaciones de significado o contexto, de secuencia lógica o de secuencia temporal, y de contenido. En una aplicación, cada una de éstas relaciones se implementa mediante links, los cuales no solamente conectan los elementos de información sino que también son usados para que el usuario navegue cada una de las entidades. En particular, los links pueden clasificarse en dos grandes grupos: *links de estructura* y *links asociativos*.
Dentro del grupo de los links de estructura se encuentran los que se refieren a la organización de la información. En un libro, por ejemplo, existe una estructura lineal que va desde el comienzo del libro hasta el final y una estructura jerárquica que define los capítulos, las secciones, párrafos, etc. Por otro lado, dentro del grupo de los links asociativos se encuentran aquellos que instancian semánticamente las relaciones entre entidades de información. El ejemplo más común que se ajusta a este tipo de clasificación son las referencias cruzadas de algunos libros donde se establecen relaciones

del tipo *"para más información de X tema ver en Y"* o los links entre páginas web. La figura 1.1 muestra los diferentes tipos de link de una aplicación hypermedia.

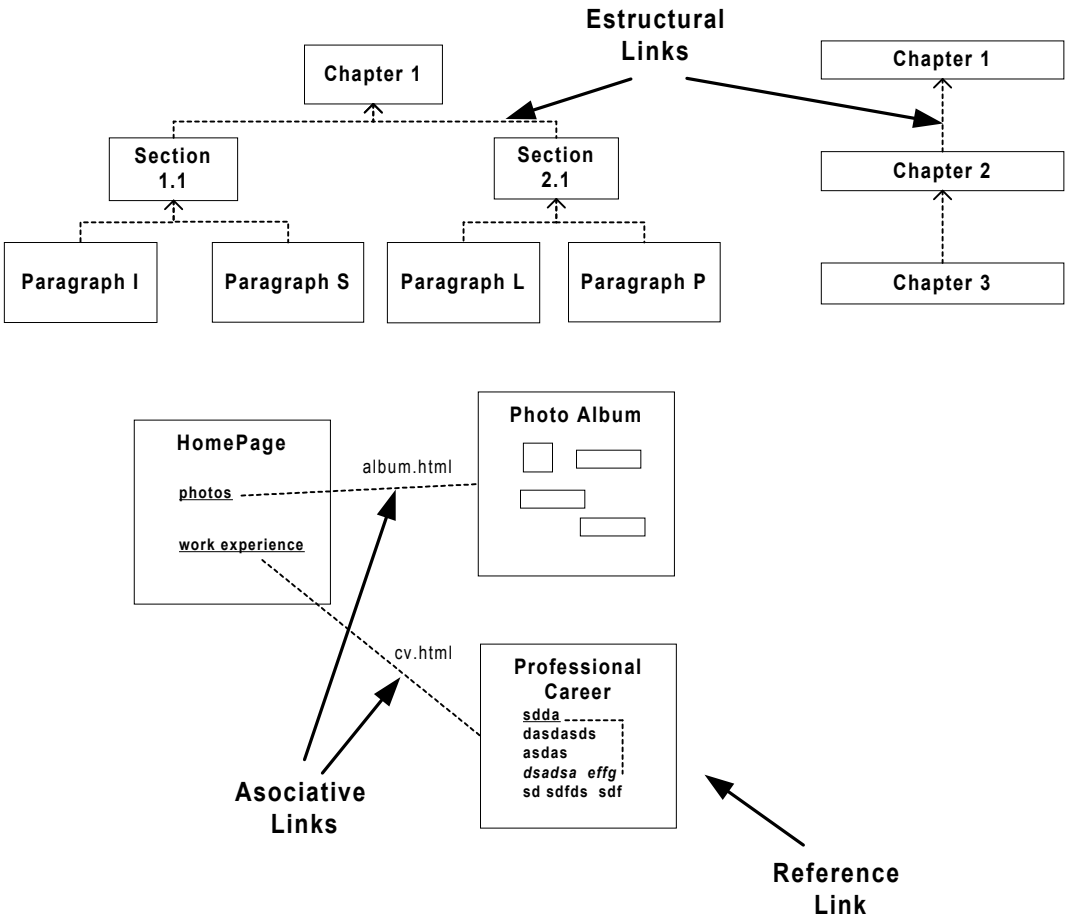

Figura 1.1: *Links estructurales, links asociativos y links de referencia.*

De ambos grupos, el primero está relacionado con la estructura del espacio de información mientras que el segundo grupo está relacionado con el contenido de la información. Existe un tercer grupo denominado *links de referencia* que se desprende del grupo de links asociativos. En este se encuentran, por ejemplo, aquellos que relacionan localmente una entidad de información con información más específica o del mismo contexto. Un ejemplo sencillo son aquellos permiten relacionar palabras con sus significados o los links locales de una página web. La diferencia que existe entre un link de referencia y un link asociativo es que en éste último las entidades relacionadas pueden existir de manera independiente mientras que en un link de referencia las entidades son mutuamente dependientes.

1.2 Aspectos Temporales de las Aplicaciones Hypermedia

Los aspectos temporales de las aplicaciones hypermedia pueden clasificarse en dos grandes grupos. Dentro del primer grupo se encuentran los que aparecen en el dominio de la aplicación. Ejemplos de estos son los que se manifiestan en las aplicaciones utilizadas para el envío de publicaciones a congresos. En este caso, los aspectos temporales están presentes al requerir que los autores envíen sus publicaciones durante fechas establecidas por la aplicación. Dentro del segundo grupo se encuentran los que están presentes en la interfaz del usuario. Un ejemplo donde éstos suceden son las interfaces gráficas de los museos virtuales. En este caso la información relacionada a una entidad principal del dominio, como por ejemplo las obras e historia de un artista, se presenta utilizando diferentes tipos de medios sincronizados en el tiempo

En la actualidad existen varios trabajos teóricos que estudian y comprenden los aspectos temporales de las aplicaciones hypermedia. Por ejemplo, en [Lue98] se hace referencia a dos tipos de tiempo: el tiempo de presentación de la interfaz del usuario y el tiempo empleado por el usuario para comprender la información presentada por las entidades. Con el mismo objetivo, en [HvOM+99] se argumenta la existencia de al menos tres puntos de vista para interpretar el tiempo: del autor, del diseñador del sistema hypermedia y del diseñador del sistema multimedia. Ambos trabajos aportan una gama de conceptos teóricos antológicamente definidos que permiten identificar el concepto de tiempo dentro de las aplicaciones hypermedia. Nosotros entendemos que durante el proceso de desarrollo de aplicaciones hypermedia, en particular aplicaciones web, es imprescindible realizar la especificación de cualquier aspecto temporal porque éstos son tan importantes como cualquier otro aspecto de la aplicación. Por tal motivo, consideramos necesario realizar la definición de mecanismos que ayuden a resolver este problema de diseño.

En la última década, los métodos de ingeniería de software han evolucionado hacia el diseño orientados a objetos proporcionando un nivel de abstracción mucho más alto que los métodos tradicionales, agregando flexibilidad al diseño, y permitiendo obtener modelos simples de extender y de mantener. Uno de los métodos que emplea este tipo de diseño es OOHDM (Object Oriented Hypermedia Design Method) [SR95b]. Este método considera el desarrollo de las aplicaciones hypermedia como un proceso basado en cuatro fases: el diseño del modelo de la aplicación, el diseño del esquema navegacional, el diseño de abstracto de la interfaz del usuario y la implementación. Esta división permite desacoplar el análisis del dominio de la aplicación en varias capas, obtener diseños más

ricos, y facilitar su implementación. Dado el gran aporte que realizan todas estas características, durante el desarrollo del presente trabajo emplearemos OOHDM como base para aplicar los modelos de diseño propuestos.

A continuación, el capítulo 2 introduce los modelos teóricos de tiempo más conocidos de la literatura. Luego, el capítulo 3 describe las características principales de algunas aplicaciones, herramientas y ambientes de desarrollo que implementan tales modelos.
El capítulo 4 da un breve repaso sobre las metodologías de desarrollo de aplicaciones hypermedia más utilizadas. Dado que este trabajo tiene como objetivo desarrollar un modelo de especificación de tiempo orientado a objetos, el capítulo 5 introduce los conceptos generales de OOHDM.
Los capítulos 6, 7 y 8 presentan los modelos de especificación de tiempo y, para terminar, el capítulo 9 muestra algunos ejemplos de aplicación.

Capítulo 2

Modelos de Tiempo

Seleccionar una adecuada representación que modele el *tiempo* durante la etapa de especificación de una aplicación hypermedia es una tarea crucial. En la actualidad se postulan diferentes clases de modelos para representar el tiempo como componente computacional. Entre las propuestas más interesantes se encuentran los modelos *Point-based*, *Interval-based*, *Axes-based*, *Control Flow-based*, *Event-based*, *Petri Nets-based*, *Duration-based* y *Scripts*. A continuación se describen las características principales, ventajas y desventajas de cada enfoque.

2.1 Modelos Point-based

Las componentes principales de los modelos *Point-based* son un conjunto de operadores temporales y eventos. Los operadores temporales se reducen a los operadores clásicos de comparación $<$, $>$, $=$ y los eventos a las acciones relevantes que forman parte de la presentación del documento. Este modelo tiene dos ventajas. La primera es que el algoritmo de comparación entre eventos es una comparación numérica de orden constante entre sus duraciones. La segunda es que para saber la duración de cada evento, sólo se necesita realizar una operación de substracción entre los valores de tiempo asignados a cualquier par de eventos consecutivos. Por otro lado, éste modelo requiere que los eventos sean marcados con una referencia temporal única. Esto último es una desventaja porque solamente es útil para especificar aspectos temporales en aquellas aplicaciones donde cada evento tiene una identificación temporal absoluta. La figura 2.1 se muestra un ejemplo de especificación.

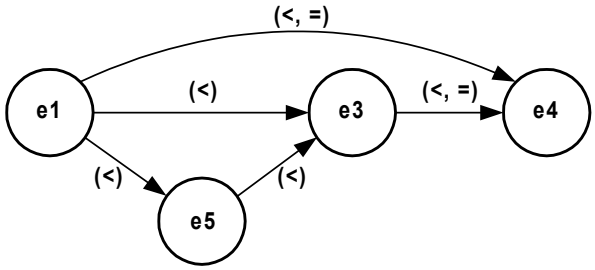

Figura 2.1: *Especificación Point-based.*

Entre las aplicaciones que emplean éste esquema se encuentran las que manejan transacciones bancarias o las que registran las transacciones de bases de datos. Un aspecto importante a tener en cuenta en éste modelo es el empleo de un algoritmo para el chequeo de consistencia de la especificación. En este sentido, en [MS90] se propone un algoritmo que utiliza un grafo sobre el cual se calcula el orden total de la especificación.

2.2 Modelos Interval-based

En los modelos *Interval-based* [All83] [Hamblis 89] [Ham72] las componentes elementales son intervalos de tiempo y un conjunto de operadores temporales. Cada intervalo modela la duración de un evento y se especifica mediante la definición de un valor inicial y valor final. Los operadores temporales son relaciones binarias que se aplican entre intervalos. Una relación puede ser paralela (during, equals, starts, overlaps, finishes) o secuencial (meets, after). La figura 2.2 resume siete relaciones definidas para este modelo de representación [LG90].

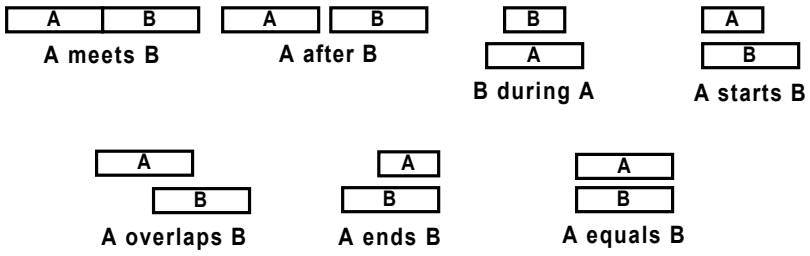

Figura 2.2: *Operadores del modelo Interval-based.*

Este modelo se desarrolló a partir de la necesidad de poder expresar razonamiento temporal y desarrollar lógicas de tiempo en el área de la inteligencia artificial. Su definición formal se especifica

en [All83] junto con todas las relaciones temporales y un algoritmo que calcula el orden total de la especificación. La desventaja que tiene este modelo es que no se pueden especificar escenarios temporales donde las entidades tienen un tiempo de presentación no predecible.

En [WR94] se propone el modelo *Enhanced Interval-based*. Este enfoque tiene sus orígenes en el modelo de relaciones de intervalos de tiempo definidas en [All83]. Las componentes principales son operadores temporales que surgen de los patrones de comportamiento detectados en las relaciones de intervalos de tiempo anteriormente descritas. La figura 2.3 muestra los operadores resultantes.

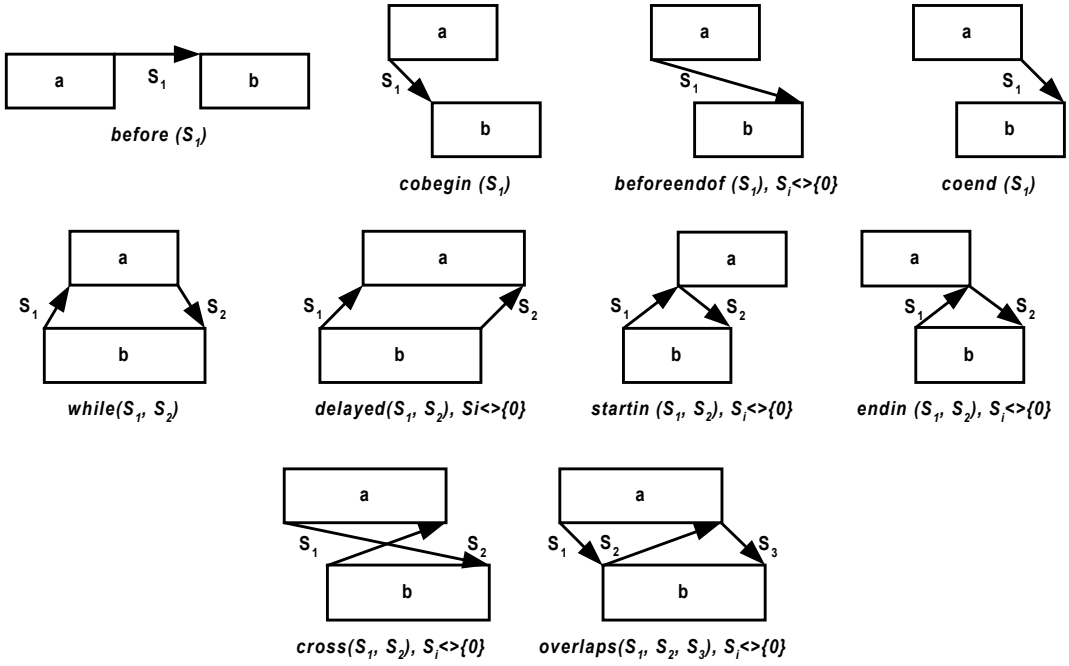

Figura 2.3: *Operadores del modelo Enhanced Interval-based.*

De acuerdo a su aridad, este desarrollo origina tres clases diferentes de operadores: operadores unarios, operadores binarios y operadores ternarios. Los operadores unarios son before, cobegin, beforeendof y coend. Los operadores binarios son while, delayed, endin y cross. El único operador ternario es overlaps. La duración de cada intervalo y el parámetro S_i se especifica con un valor mayor o igual a cero. En la figura 2.4 se muestra un ejemplo de uso.

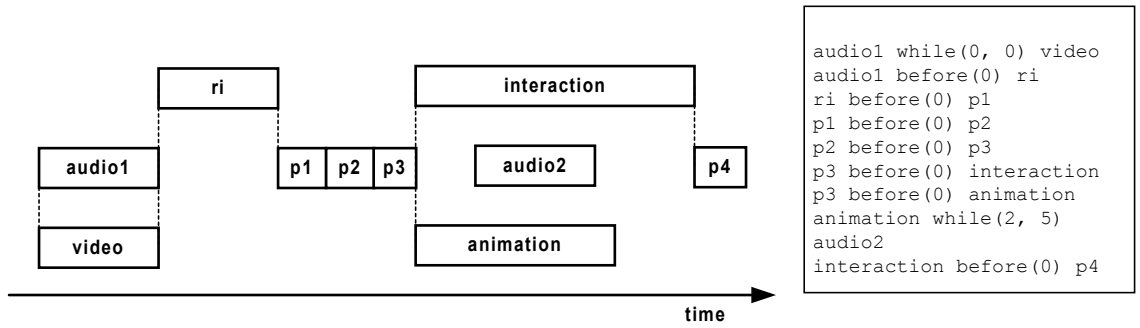

Figura 2.4: *Especificación Enhanced Interval-based.*

La ventaja que provee este modelo es que se pueden especificar interacciones del usuario y entidades cuyo tiempo de presentación son no predecibles. Sin embargo, la especificación resultante es compleja de entender y pueden suceder inconsistencias en tiempo de ejecución [BS96]

En [DK96] se propone un modelo denominado *Interval Expressions*. Este enfoque se basa en los conceptos desarrollados en [WDG95]. Las entidades que se emplean se denominan media segments y operadores temporales.
Un media segment es un objeto multimedia tipado con comportamiento temporal inherente. Los tipos definidos para un media segment son cuatro: multimedia tradicionales, streams multimedia, programas o links. Los dos primeros engloban los media segment que no reciben ningún tipo de interacción y los dos últimos los que pueden recibir interacción por parte del usuario o de otro media segment. Los media segments de tipo multimedia tradicionales presentan video, audio, texto o imagen, los de tipo streams multimedia reproducen streams de teleconferencia, radio o televisión, los del tipo programa contienen bloques de código ejecutable por el usuario y los de tipo link referencian a otros media segment que inician su presentación al activarse el link que los referencia.

Para realizar la especificación del aspecto temporal de una aplicación, el autor propone interpretar cada media segment como un intervalo de tiempo $a = \{t | a \leq t \leq pf\}$ y emplearlo como argumento de los operadores de intervalos tiempo. Los operadores tienen origen a partir de las relaciones que pueden suceder entre dos intervalos de tiempo. La figura 2.5 muestra estas relaciones.

Un operador temporal puede tener como argumentos uno o muchos intervalos de tiempo y su resultado es otro intervalo de tiempo. Los operadores denotan composiciones temporales dependiendo

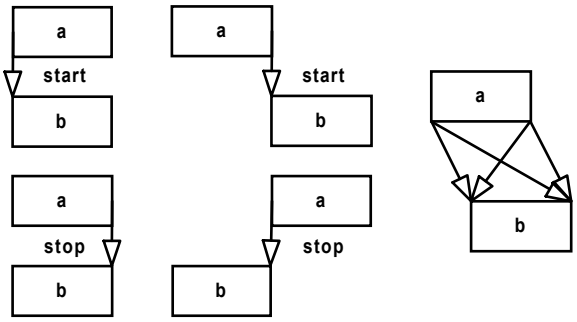

Figura 2.5: *Relaciones entre intervalos de tiempo.*

de la cantidad de sus argumentos y de la forma en que se define cada operador. Los operadores se definen de la siguiente manera:

seq(a, b)→(a, b)	define una composición donde el fin del intervalo *a* da inicio al intervalo *b*.
follow(a, b)→(a, b)	define una composición donde el inicio del intervalo *b* da fin al intervalo *a*. El intervalo *b* es activado por el usuario.
par-begin(a, b)→(a, a)	define una composición donde el inicio del intervalo *a* da inicio simultáneo al intervalo *b*
par-end(a, b)→(a, a)	define una composición donde el fin del intervalo *a* da fin al intervalo *b*. El intervalo *b* es activado por el usuario.
par-min(a, b)→(a, min(a, b))	define una composición donde el inicio del intervalo *a* da inicio al intervalo *b*. El interval resultante finaliza con el fin del intervalo más corto.
par-max(a, b)→(a, max(a, b))	define una composición donde el inicio del intervalo *a* da inicio al intervalo *b*. El interval resultante finaliza con el fin del intervalo más largo.
equal(a, b)→(a, a)	define una composición donde el inicio del intervalo *a* da inicio y fin al intervalo *b*.
ident(a, b)→(a, b)	define una composición donde el inicio del intervalo *a* da inicio al intervalo *b* y el fin del intervalo *b* da fin al intervalo *a*.

Figura 2.6: *Operadores del modelo Interval Expressions.*

Una de las ventajas que presenta este modelo es que el anidamiento de operadores se puede realizar fácilmente. Esto permite especificar presentaciones complejas como por ejemplo la que se muestra en la figura 2.7 a.

Sin embargo, la composición de intervalos de tiempo a través de operadores no garantiza la especificación de cualquier escenario temporal. Por ejemplo, la figura 2.7 b muestra un escenario que no puede ser especificado con este enfoque. En otras palabras, la composición de intervalos está

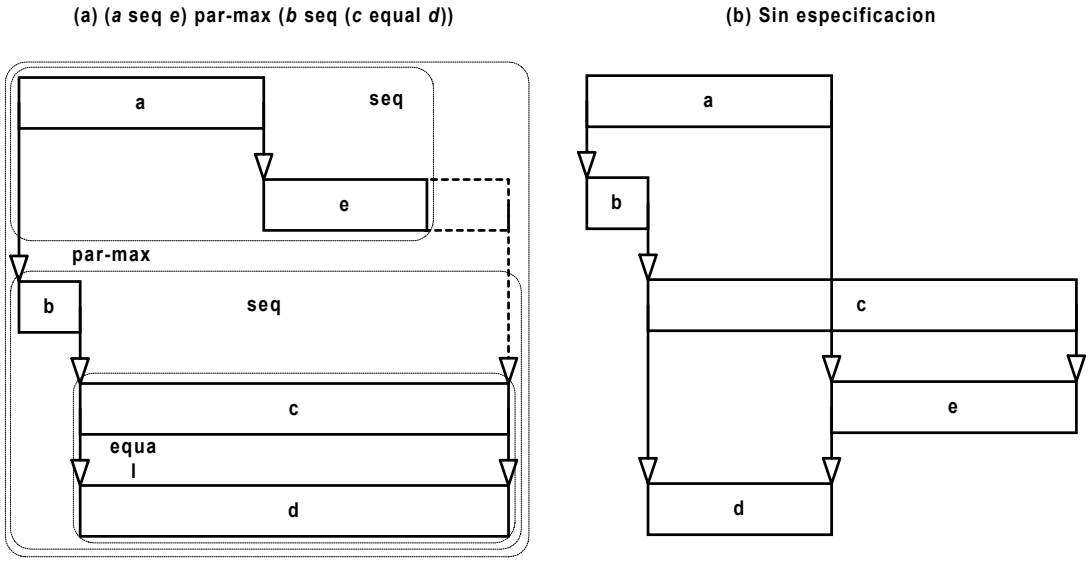

Figura 2.7: *Especificación Interval Expressions.*

restringida a la clase de escenarios temporales que pueden ser especificados. *Temporal Nets* [BZ93] [ISO92] es uno de lo modelo apropiado para realizar ese tipo de especificaciones, sin embargo la complejidad del modelo resultante no es fácil de mantener.

2.3 Modelos Axed-based

La lógica de los modelos *Axes-based* consiste en especificar la ocurrencia de eventos a lo largo de ejes temporales. En otras palabras, cada evento que se considera importante para la presentación de un objeto se proyecta sobre ejes de tiempo. Existen dos variantes: *global timer* y *virtual axes.*

En la variante *global timer* la duración de cada entidad que forma parte de una presentación se proyecta sobre un único eje de tiempo. El eje es una representación abstracta del tiempo real que dura toda la presentación. Las proyecciones de las duraciones son totalmente independientes una de la otra. En la figura 2.8 se muestra un ejemplo.
El hecho de realizar la proyección de la duración de cada una de las entidades sobre un eje de tiempo permite abstraerse de la estructura interna de cada entidad. Por ejemplo, para especificar la presentación simultánea de los subtítulos de un video y la proyección del mismo no se requiere conocer la estructura interna de las entidades. Sin embargo, dado que la sincronización se realiza entre puntos fijos, podrían surgir problemas si alguna de las entidades posee un tiempo de duración no

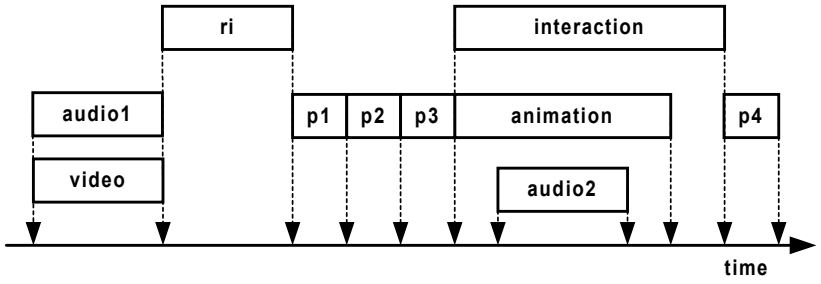

Figura 2.8: *Especificación Global Timer.*

predecible. Además, es importante tener en cuenta que ésta variante obliga que la duración de cada presentación sea dependiente del eje de tiempo. Esto puede provocar problemas en la presentación de todas las entidades en caso que el tiempo de presentación de alguna de ellas se retrase o se prolongue.

La variante *virtual axes* es una generalización de la variante global timer. En este enfoque está permitido definir la cantidad de ejes de tiempo que sean necesarios. La figura 2.9 muestra un ejemplo donde se especifica un eje de tiempo para los objetos que requieren interacción del usuario y otro eje de tiempo para los demás objetos. Este modelo posee las mismas ventajas que el anterior.

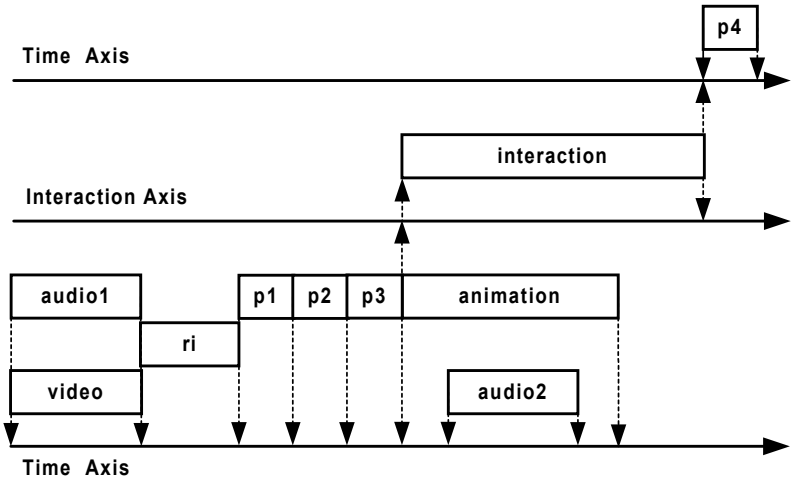

Figura 2.9: *Especificación Virtual Axes.*

Además, dado que está permitido definir diferentes ejes de sincronización, quedan resueltos todos los problemas de entidades con duración no predecible. Sin embargo, el resultado que se obtiene es un modelo mucho más complejo y la interpretación de los ejes de tiempo se convierte en un mecanismo costoso de implementar [BS96].

2.4 Modelos Control Flow-based

La lógica de este tipo de modelos consiste en definir puntos de sincronización que coordinen la presentación de entidades multimedia. Existen tres variantes: *hierarchical specification* y *reference points*.

Las entidades trascendentales del modelo *hierarchical specification* [SS90] [Gro89] son acciones, operaciones de sincronización y árboles. Una acción puede ser de tipo atómica o compuesta. Las acciones atómicas modelan la presentación de cualquier objeto multimedia o interacción del usuario mientras que las acciones compuestas especifican una combinación de operaciones de sincronización y acciones atómicas. Una operación de sincronización establece si dos acciones ocurren de manera secuencial o paralela. En un árbol una hoja representa una acción atómica y cada nodo una acción compuesta. La figura 2.10 muestra un ejemplo de especificación de este modelo.

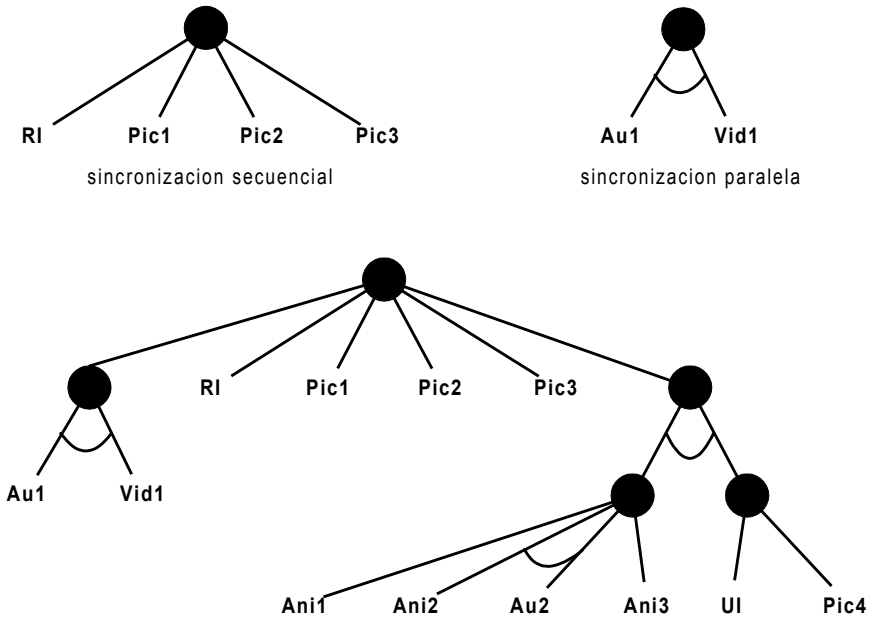

Figura 2.10: *Especificación Hierarchical Specification.*

Este modelo es ampliamente usado porque es fácil de emplear y porque soporta cualquier tipo de presentaciones que tengan duración indeterminada. Sin embargo, la gran desventaja radica en que un grupo de acciones solamente pueden ser sincronizadas en sus puntos limites. Por ejemplo, no se puede especificar el mismo escenario que especifica overlaps del modelo de intervalos de tiempo.

Por su parte, *reference points* [BHM92] [Ste90] define y emplea el concepto de puntos de referencia para especificar escenarios temporales entre entidades. Un punto de referencia es un punto que señala un instante inicial o final de la presentación de una entidad y puede ser de tipo interno o externo. Un punto de referencia interno es un punto que marca los límites de la presentación de una unidad lógica [BS96]. Un punto de referencia externo es un punto que marca los límites de la presentación de una entidad multimedia integrada por unidades lógicas.

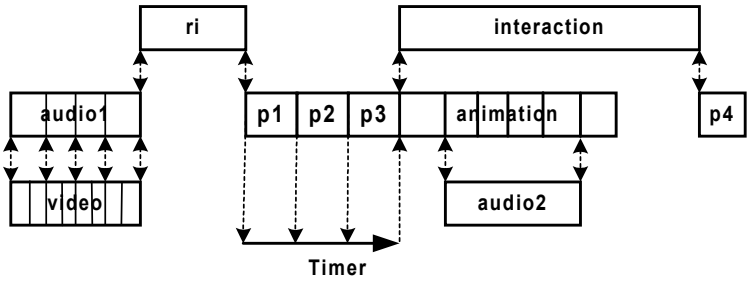

Figura 2.11: *Especificación Reference Point.*

La especificación del aspecto temporal se realiza conectando los puntos de referencia de las entidades involucradas. Un conjunto de puntos de referencia conectados se denomina punto de sincronización. La presentación de las unidades lógicas que participan en el mismo punto de sincronización debe comenzar o finalizar cuando el punto de sincronización es alcanzado. La figura 2.11 muestra un ejemplo de especificación empleando este enfoque de especificación.

En las especificaciones que emplean este modelo, las relaciones temporales entre elementos multimedia quedan implícitas. En otras palabras, no se necesita emplear ninguna referencia a alguna unidad o eje de tiempo. Otra ventaja es que la presentación de una entidad multimedia con tiempo de presentación no predecible puede ser integrada fácilmente. Una de las desventajas que se presentan es que se necesita emplear mecanismos externos para detectar inconsistencias en la especificación. Además, este modelo no provee ninguna forma de especificar demoras en la presentación. En [Ste90] se propone una solución a este problema. Asimismo, dada la descripción directa de relaciones entre entidades, este modelo resulta difícil de mantener.

2.5 Modelos Event-based y Scritps

El modelo *event-based* consiste en definir una grilla con el orden en que serán realizadas las acciones de presentación de las entidades multimedia. Una acción de presentación se inicia a causa de la ejecución uno o varios eventos. Estos eventos pueden ser internos, como por ejemplo la finalización de una secuencia de video, o externos, como por ejemplo la señal de un timer. La figura 2.12 muestra un ejemplo de especificación.

Event Action	Start	Audio1.stop	Timer1.ready
Audio1	start		
Video	start		
Pic.1		start	stop
Timer1		Start(3)	
Pic.2			start

Figura 2.12: *Especificación Event-based.*

En este modelo es fácil integrar entidades interactivas. Además, es simple de extender en caso que sea necesario agregar nuevos eventos. Sin embargo, resulta difícil de manejar al especificar escenarios temporales reales. Asimismo, la especificación resultante es compleja, ardua de mantener y para la integración de elementos multimedia dependientes del tiempo es necesario emplear timers adicionales.

Los modelos de *scripts* son descripciones textuales que indican la manera en que deben suceder las presentaciones de las entidades multimedia participantes [Cor90] [TGD91] [Gib91]. Los elementos principales de un script son actividades y otros scripts. Por lo general, los scripts terminan convirtiéndose en código de algún lenguaje de programación que importan librerías con rutinas de tiempo. Los scripts soportan tres operadores principales: presentación secuencial, presentación paralela e iteración de la presentación. En la figura 2.13 se muestra un ejemplo.

Este modelo permite integrar entidades dependientes del tiempo y objetos interactivos. La principal desventaja es que en la mayoría de sus aspectos este método es totalmente estructurado y poco declarativo, lo cual dificulta su empleo porque requiere que los autores tengan nociones de programación.

```
activity DigAudio          Audio ("video.au");
activity SMP               Video ("video.smp");
activity Xrecorder         Recorder("window.rec");
activity Picture           Pic1("picture1.jpeg");
activity Picture           Pic2("picture2.jpeg");
activity Picture           Pic3("picture3.jpeg");
activity Picture           Pic4("picture4.jpeg");
activity StartInteraction  Selection;
activity DigAudio          AniAudio("animation.au");
activity RTAnima           animation("animation.ani");
script Picture_sequence
       3Pictures  Pic1.Duration(5) >> Pic2.Duration(5) >> Pic3.Duration(5);
script Lipsynch AV = Audio & Video;
script AniComment AA = Animation & AniAudio.Translate(2);
script Multimedia
       Application_example {
                          AV >> Record UI >> 3Pictures >> ((Selection Pic4) && AA)
       }
```

Figura 2.13: *Especificación Script.*

2.6 Modelos Petri Nets-based

En [SWD95] se propone *HTSPN (Hierarchical Time Stream Petri Nets)* como una extensión al modelo Petri Nets [S95b]. Las componentes principales que componen este modelo se denominan temporal validity intervals. Un temporal validity interval es una tripla (x, y, n) donde x es el valor mínimo, n el valor nominal e y el valor máximo de tiempo admisible que puede durar el procesamiento de una entidad asociada. Cada temporal validity interval se adjunta, como etiqueta, sobre un arco originando así el concepto de arco temporizado. Este mecanismo permite modelar, analizar, verificar y simular formalmente una aplicación multimedia distribuida [S95a].

Los mecanismos de especificación que HTSPN provee son componentes atómicas, componentes compuestas y links. Una componente atómica es la representación abstracta de un nodo y se simboliza mediante un place de tipo atómico asociado a un arco etiquetado con un temporal validity interval. Una componente compuesta es una estructura jerárquica basada en la composición recursiva de componentes atómicas y otras compuestas, y se simboliza de la misma manera que una componente atómica. Un link es un enlace entre componentes compuestas o atómicas y se simboliza mediante un place de tipo link y un arco etiquetado con un temporal validity interval. Este último mecanismo

introduce el concepto de link temporizado. La figura 2.14 muestra un ejemplo de especificación HTSPN.

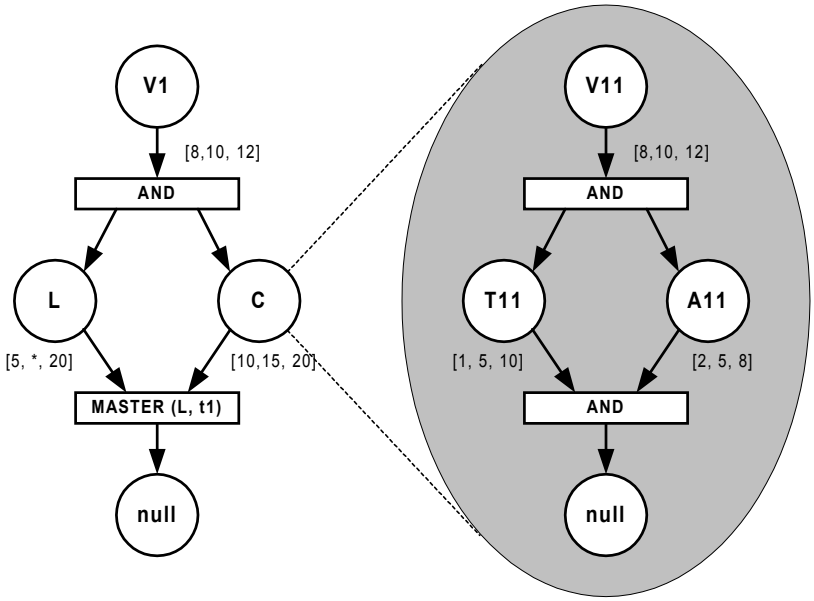

Figura 2.14: *Especificación HTSPN.*

El modelo HTSPN provee tres capas de especificación: link synchronization layer, composite synchronization layer y atomic synchronization layer. En la primer capa se especifican los mecanismos de navegación de la aplicación, en la segunda los escenarios multimedia de un documento y en la tercer capa cada una de las entidades multimedia.

El modelo *Timed Petri Nets* [LG91] [LG92] extiende la especificación de Petri Nets agregando reglas especificas para la especificación de tiempo. Las reglas de una timed petri net son las siguientes:

- Una transición se ejecuta si no contiene un token no bloqueante.
- Si una transición se ejecuta, se elimina un token de cada input place y se agrega un token a cada output place.
- Un token que es agregado a un place se bloquea mientras dure ese place.

La figura 2.15 muestra un ejemplo de especificación.

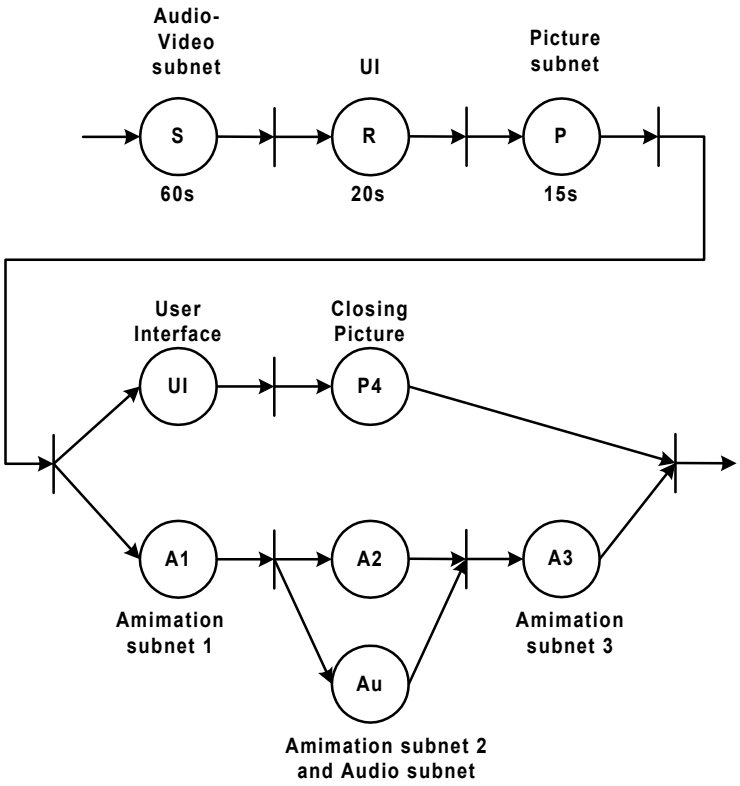

Figura 2.15: *Especificación Timed Petri Net.*

Este modelo soporta todo tipo de especificación temporal incluyendo presentaciones con duración indeterminada. Sin embargo, posee dos desventajas principales: la falta de abstracción de las entidades multimedia y la complejidad de modelo resultante.

2.7 Modelos Duration-based

Los modelos *Duration-based* tienen sus orígenes en el área de las redes de PERT. El modelo de representación que se utiliza es un grafo dirigido sin ciclos. Cada nodo modela un evento y su etiqueta especifica un intervalo de tiempo durante el cual un evento puede empezar a ejecutarse. Por su parte, la dirección de cada arista establece el orden en que dos eventos suceden y su etiqueta señala la duración entre ambos. En este esquema se destacan un nodo inicial y otro final que señalan el comienzo y el fin de la ejecución de una serie de eventos. La restricción que impone este modelo es que se necesita saber por anticipado la duración de cada evento. Por tal razón, es ampliamente utilizado en el scheduling de aplicaciones. La figura 2.16 muestra un ejemplo de especificación.

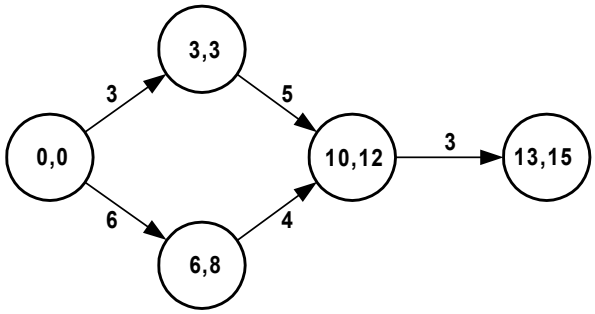

Figura 2.16: *Especificación Duration-based.*

En [DM87] se propone el modelo de un esquema similar. En ese caso, la representación que se propone codifica la información temporal en el dígrafo empleando el concepto de temporal constraints. La diferencia consiste en etiquetar las aristas con la duración entre dos eventos en lugar de emplear la duración de un evento.

Capítulo 3

Implementaciones con Modelos de Tiempo

La sección anterior resume las características, ventajas y desventajas de los principales enfoques existentes para especificar aspectos temporales en una aplicación hypermedia. Esta sección describe algunas aplicaciones, herramientas de diseño, estándares y frameworks más utilizados que implementan esos modelos; entre estos se encuentran *Firefly*, *MHEG*, *HyTime*, *MODE*, *Littles Framework*, *MADEUS* y *Temporal HTML*. La mayoría se componen de varias capas o niveles que incluyen especificación, implementación, manejo de errores, transferencia de datos, persistencia y presentación. Dado que el enfoque de este trabajo esta orientado hacia la especificación del tiempo en el diseño de aplicaciones hypermedia, solamente se presentarán los detalles del nivel de especificación y algunas características básicas del nivel de implementación. En el nivel de especificación se emplean algunos de los enfoques de especificación descritos en la sección anterior, mientras que en la implementación se utilizan mecanismos específicos de sincronización de manera que la implementación concuerde con lo especificado.

3.1 Firefly Document System

Firefly [BZ93] [MP93] es una aplicación que sirve para diseñar, desarrollar y presentar aplicaciones hypermedia incluyendo los aspectos temporales definidos entre las componentes de la interfaz del usuario. Esta aplicación se basa en la especificación y composición de tres entidades: media items, temporal synchronization constraints y operation lists.

Un media item describe el comportamiento temporal de cualquier entidad de información que pueda ser presentada en un documento hypermedia. Todo media item se compone de cuatro partes principales: un media type, eventos, procedimientos y un puntero. El media type especifica el tipo de medio que el media item representa como por ejemplo un video, una fracción de audio, texto, imágenes, etc. Los eventos modelan los puntos en los cuales la presentación del media item puede ser sincronizada con las de otros media items y se dividen en dos tipos: sincrónicos y asincrónicos. Los eventos sincrónicos son aquellos de los cuales se conoce el tiempo de duración, mientras que los eventos asincrónicos son aquellos eventos cuyo tiempo de duración no puede ser determinado hasta que el media item sea presentado. Cada media item posee dos eventos elementales que señalan el inicio y el fin de su presentación. Un procedimiento es una acción que realiza operaciones entre un media item y sus eventos. Los procedimientos pueden ser de tres tipos: de especificación de comportamiento, de análisis de tiempo y de control. Los procedimientos de especificación de comportamiento dan acceso, a través de la interfaz del usuario, a la creación y edición de eventos del media item. Los procedimientos de análisis de tiempo proveen información de tiempo del media item al sistema para la creación del scheduling de un documento. Los procedimientos de control permiten que Firefly controle el comportamiento de un media item mientras el documento esta siendo presentado. Por ultimo, el puntero es una referencia lógica a los datos del media item.

Para especificar los aspectos temporales de un documento, los métodos que se utilizan son del tipo event-based empleando temporal synchronization constraints. Un temporal synchronization constraint es una especificación de tiempo entre dos eventos. Firefly admite dos tipos de temporal synchronization constraints: temporal equalities y temporal inequalities. Temporal equalities son restricciones temporales que especifican que dos eventos sucedan simultáneamente o que uno suceda antes que otro con una diferencia de tiempo fija. Mientras que temporal inequalities son restricciones temporales que especifican que un evento suceda antes que otro dentro de un rango de tiempo o bien que un evento suceda en algún momento antes que otro evento. En otras palabras el primer tipo de restricción especifica sincronización concreta y el segundo tipo especifica sincronización tentativa.

Las operation lists controlan la presentación de un documento. Cada operation list se compone de dos partes: una lista ordenada de operaciones que pueden o no modificar el tiempo de presentación entre dos eventos y un puntero al evento con el cual la lista esta asociada.
Firelfy se compone de un mecanismo para especificar documentos construyendo grafos. Cada nodo

de un grafo representa un evento en un media item y cada arista un temporal synchronization constraint. Además, para la edición de media items, Firefly se adapta al empleo de algunos editores de texto, imágenes, video y sonido del mercado. El scheduling consiste de una lista ordenada de comandos que el sistema utiliza para controlar la presentación de los media items.

3.2 MHEG

MHEG (Multimedia and Hypermedia Expert Group) [Ech98] define un formato estándar para la representación y codificación de entidades multimedia y hypermedia. La definición del estándar se compone de un conjunto de clases de objetos predefinidas, un ambiente de ejecución, un espacio de coordenadas virtual y un eje de tiempo. Entre las clases de objetos predefinidos más importantes se encuentran: Content, Composite, Action, Link y Script. El espacio de coordenadas virtual que especifica la posición de las entidades multimedia dentro del documento y se compone de un eje de latitud, un eje de longitud y un eje de altura (X, Y, Z). La especificación de los aspectos temporales de un documento se lleva a cabo empleando métodos axes-based o event-based.

Para especificar escenarios temporales empleando un enfoque axed-based se proyectan todos los objetos que forman parte de la presentación sobre el eje de tiempo. La unidad de medida que emplea este eje se denomina GTU (Generic Time Unit). Para cualquier aplicación que se implemente empleando MHEG, el diseñador especifica si quiere mapear cada unidad GTU a una unidad PTU (Physical Time Unit) o a un milisegundo. De otro modo, para especificar escenarios temporales empleando un enfoque event-based, se utilizan instancias de la clase Link, condiciones y eventos. La lógica de esta alternativa es activar el link y ejecutar sus acciones asociadas cuando la condición se cumple.

3.3 HyTime

HyTime [ISO92] es una aplicación SGML (Standarized General Markup Language) [65] que define un formato estándar para representar entidades hypermedia de manera estructurada. Contrariamente a lo que sucede en MHEG, HyTime no codifica ni formatea ninguna entidad multimedia. HyTime provee un framework para especificar relaciones entre las entidades que componen un documento hypermedia. Además, incluye mecanismos que sirven para identificar puntos internos de una entidad, facilidades para conectar entidades mediante links y especificaciones para describir relaciones

de espacio y tiempo entre las entidades que forman un documento.

HyTime define la semántica de seis módulos diferentes necesarios para realizar la implementación de una aplicación hypermedia. Estos módulos son: Base Module, Measurement Module, Location Address Module, Scheduling Module, Hyperlink Module y Rendition Module.

Para especificar los aspectos temporales de un documento, el método que se utiliza es del tipo axes-based y el módulo donde se especifican estas características es el Scheduling Module. Este módulo posiciona las entidades multimedia que forman parte de la presentación en espacios de coordenadas de dimensión finita (FCS). Cada uno de estos espacios se compone de una colección de ejes de tiempo que se especifican en la etapa de diseño de la aplicación. Las unidades de medida con que trabajan los ejes se definen en el Measurement Module.
HyTime no manipula directamente las entidades multimedia que componen el documento. En su lugar, se emplean otras entidades denominadas eventos que encapsulan entidades multimedia y especifican las propiedades necesarias para realizar su presentación y localización dentro de un (FCS).

3.4 MODE

MODE (Multimedia Objects in a Distributed Environment) [Bla93] es una aplicación desarrollada en la Universidad de Karlsrühe. MODE provee un servicio distribuido de presentaciones multimedia que comparte un modelo multimedia y un grupo de especificaciones de sincronización. Además, por ser una aplicación distribuida, también contiene información referida a redes y unidades remotas. Las componentes principales de MODE son las siguientes: Synchronization Editor, MODE Server Manager, Local Synchronizer y Optimizer.
El método que se utiliza para especificar los aspectos temporales de un documento es del tipo reference point-based extendido [BS96]. Las extensiones agregadas permiten utilizar intervalos de tiempo y especificar entidades multimedia con tiempo de duración.

La especificación del aspecto temporal de las entidades multimedia que conforman un documento especificado con MODE se realiza en el Synchronization Editor y se almacena en formato textual escrito en un lenguaje denominado Synchronization Description Language. Dado el modelo de especificación subyacente, MODE hace diferencia entre la presentación de objetos dinámicos y la presentación de objetos estáticos. En la presentación de entidades multimedia dinámicas, éstas son

consideradas como una composición secuencial de unidades lógicas y el índice de cada presentación es un punto de referencia, mientras que en la presentación de entidades estáticas el índice son dos puntos de referencia.
La componente Local Synchronizer realiza las presentaciones de acuerdo a lo especificado en el Synchronization Editor empleando threads y un esquema de prioridades. La sincronización entre los threads se realiza utilizando mecanismos de señalización.

3.5 Littles Framework

Este framework fue desarrollado en la Universidad de Boston [LG90] [LPL95] y se compone de cuatro métodos principales: un método para la especificación de sincronización, un método para la representación de entidades multimedia, un método para el control de acceso temporal y un método para la sincronización de entidades multimedia en tiempo de ejecución. Además, permite realizar la presentación de documentos compuestos por entidades multimedia locales o remotas.

Los métodos de especificación de los aspectos temporales que utiliza el framework son del tipo petri nets-based y global timer-based. Para administrar las entidades que componen el documento y realizar su presentación, el framework utiliza mecanismos de scheduling estático y dinámico. Además, el scheduling estático junto con otros dispositivos de transferencia de datos son empleados para resolver, por ejemplo, problemas causados por el tipo de almacenamiento, comunicación o procesamiento de entidades locales o remotas. Por su parte, el scheduling dinámico también se encarga de resolver problemas que surgen a causa de cambios en el entorno de presentación o la ejecución de comandos que alteran, por ejemplo, la velocidad de la presentación de un video.

3.6 MADEUS

MADEUS [JL$^+$98] es una herramienta desarrollada para diseñar documentos multimedia. Un documento MADEUS se compone de un conjunto de tags que describen entidades multimedia y relaciones temporales [Lay96]. Para especificar los aspectos temporales de un documento, el método que se utiliza en la etapa de especificación es del tipo temporal constraint networks [DMP91] extendido. Las extensiones agregadas por MADEUS son: additional labelling, causality relations, spatio-temporal actions y hyperlink basic objects.

Additional labelling agrega semántica a las aristas para que modelen presentaciones con tiempo no predecible o predecible. Causality relations es una nueva propiedad agregada a los nodos para especificar relaciones del tipo master-slave. Spatio-temporal actions es un conjunto de propiedades y acciones que ayudan a especificar los instantes en que un nodo puede aparecer o desaparecer, durante la presentación del documento, independientemente de la duración de su presentación. Los objetos del tipo hyperlink permiten enlazar diferentes presentaciones multimedia. La manera en que estos se agregan al método es como intervalos de tiempo modelando así el periodo de tiempo durante el cual están disponibles.

La arquitectura de MADEUS se compone de cuatro niveles diferentes: specification level, scheduler level, execution mediator level y object level. Specification level provee al sistema un lenguaje de especificación simple y altamente expresivo para especificar relaciones temporales necesarias entre entidades multimedia. Además, este nivel proporciona un parser y un translator. El primero convierte la especificación en una representación interna de la herramienta sobre la cual se realizan chequeos de consistencia de la especificación. El translator convierte la representación resultante del parser en un formato apto para ser almacenado. Scheduler level maneja la sincronización especificada entre las entidades multimedia del documento. Execution mediator level entrega los estados de las entidades multimedia al scheduler level como por ejemplo el tiempo de presentación. Object level maneja las acciones que reciben y ejecutan cada entidad multimedia. En [JL+98] se da una descripción más detallada de cada uno de los niveles que forman parte de esta aplicación.

3.7 HTML Extendido

En [RD97]se define *Temporal Extensions to HTML* para especificar y reproducir presentaciones multimedia sincronizadas en la web. La especificación de los aspectos temporales de un documento se realiza empleando un paradigma funcional derivado del modelo temporal point nets. La extensión que se realiza a HTML consiste en agregar hypertime links, time bases y dynamic layouts.

Un hypertime link es un enlace temporal que liga, como cualquier link, un origen con un destino con la diferencia de que este nuevo tipo de link posee una semántica temporal explícita: relaciona dos entidades multimedia y asigna el instante de tiempo del nodo origen al nodo destino. Un hypertime link puede iniciar o finalizar la presentación de una entidad multimedia. La activación de un hypertime link es implícita en el sentido que no se requiere la interacción por parte del usuario.

Un time base es un espacio de tiempo que contiene todas las entidades que están afectadas por una misma relación. En otras palabras, un time base especifica un conjunto de entidades a sincronizar, cual es el tipo de sincronización que existe entre ellas y de que manera se lleva a cabo tal sincronización. Un time base puede ser visto como un espacio de tiempo en el cual las entidades multimedia se comportan de manera ideal. Dado que tal escenario no existe, un time base emplea dos formas posibles de sincronización entre entidades multimedia: master-slave y synchronization points. Por último, un dynamic layout se define como una entidad multimedia que encapsula frames (regiones de un documento). Su responsabilidad es definir el comportamiento temporal del layout de la aplicación. Al igual que en HTML, los frames pueden estar junto con otras entidades e inclusive contener otros frames, definiendo así espacios con comportamiento temporal anidados.

Para respaldar el modelo recientemente descrito en algunos trabajos se define una arquitectura compuesta por tres partes: synchronizable objects, synchronization events y synchronization managers. La primer componente implementa entidades multimedia con funcionalidades de sincronización. La segunda componente realiza la implementación de hypertime links. Y la última componente implementa time bases, es decir que maneja un conjunto de sychronizable objects que pertenecen al mismo time base. En [Rousseau98] se describe detalladamente cada una de las componentes de la arquitectura.

Capítulo 4

Diseño de Aplicaciones Hypermedia

Existen diversas metodologías y modelos que pueden ser empleados para el desarrollo de aplicaciones hypermedia. La mayoría utiliza técnicas de diseño entidad relación o principios de diseño orientado a objetos. Entre las que emplean modelos entidad relación las más importantes son HDM (Hypermedia Design Model) [GPS93] y RMM (Relationship Management Methodology) [ISB91]. Entre las que emplean principios de diseño orientado a objetos las más importantes son EORM (Enhanced Object-Relationship Model) [Lan94] y OOHDM (Object Oriented Hypermedia Design Method) [SR95b]. A continuación se describen las características y los aspectos principales de cada una de ellas.

4.1 HDM

Hypermedia Design Model tiene sus fundamentos en el modelo de diseño E-R, extiende el concepto de entidad e introduce las primitivas unidad y link. A diferencia de las entidades usadas en el modelo E-R, las entidades de HDM poseen una estructura interna y semántica.

Una entidad se define como una jerarquía de componentes que se componen de unidades. La semántica de una entidad indica la manera en que sucederá la navegación y la forma en que se visualizará la información contenida en la entidad. Las entidades se dividen en dos tipos: entidades de aplicación y entidades de outline. Una de las diferencias que existe entre ambas es que las entidades de outline sirven para proveer acceso a las entidades de aplicación. En HDM existen tres

tipos diferentes de links: links de estructura, links de perspectiva y links de aplicación. Los links de estructura se utilizan para conectar componentes, los links de aplicación para conectar entidades, y los links de perspectiva para conectar entidades y componentes de cualquier tipo.

HDM no es una metodología de diseño, sino más bien un modelo de diseño que puede ser utilizado para describir y analizar aplicaciones hypermedia. En otras palabras, HDM provee un framework para el desarrollo de aplicaciones hypermedia pero no especifica ningún proceso de diseño.

4.2 RMM

Relationship Management Methodology fue desarrollado por un grupo de investigadores entre que se encuentran Isakowitz, Stohr, Balasubramanian y Diaz. El resultado de este trabajo de investigación es una metodología para el diseño de aplicaciones hypermedia que se basa en la combinación del modelo entidad-relación y algunos conceptos de HDM y RMDM ligeramente adaptados. Esta metodología especifica el desarrollo de una aplicación hypermedia en siete etapas: E-R design, slice design, navigational design, protocol conversion design, user interface design, run-time behavior design, y construction and testing.

La primer etapa consiste en identificar las entidades y relaciones existentes en el dominio de la aplicación. El resultado es un esquema entidad-relación y su desarrollo consiste en identificar la información y la estructura inherente del dominio de la aplicación. En la segunda etapa se establecen dos aspectos de la información contenida en la aplicación: su modo de presentación y la forma de acceso. El primero define la manera en que la información de la aplicación será presentada al usuario, mientras que el segundo aspecto define la forma en que la información será accedida. Para lograr ambos objetivos se emplean slice (entidades de RMDM) que sirven para agrupar los atributos de las entidades que serán presentados. En el diseño navegacional de RMM se especifican los caminos que sean necesarios para poder recorrer las entidades del esquema de la primer etapa. La especificación de estos caminos se realiza mediante el uso de las primitivas de RMDM: link, grouping (menues), index, guided tour e indexed guided tour. En la cuarta etapa, se define un conjunto de reglas de conversión para transformar cada elemento de RMDM en un objeto concreto del ambiente de implementación. Por ejemplo, si se requiere implementar la aplicación en la plataforma web entonces debe ser definido un conjunto de reglas que traduzca cada slice a documentos HTML. En la quinta fase, cada objeto que aparece en el esquema RMDM es utilizado como base para el diseño

de las componentes visuales de la interfaz del usuario. Se definen algunos aspectos como la visualización de las anclas, visualización de los botones, posicionamiento de videos e imágenes, etc. En la anteúltima etapa se considera la funcionalidad que implementará el comportamiento de la aplicación en tiempo de ejecución. Si por ejemplo se está desarrollando un website se deben tomar en cuenta algunos aspectos como el historial de páginas, la inclusión de motores de búsqueda, el empleo de páginas estáticas o páginas dinámicas, etc. En la última etapa se implementa toda la aplicación y se realizan los test necesarios. En este aspecto, RMM no da ninguna especificación al respecto por ser totalmente dependiente de la tecnología y la plataforma a emplear.

En la actualidad existen herramientas case que soportan los pasos especificados por RMM. Entre ellas se encuentran RM-Case (Relation Management Case Tool) [DIMG95] utilizada para realizar el desarrollo de aplicaciones web.

4.3 EORM

Enhanced Object-Relationship Model es un proceso de desarrollo iterativo que utiliza las primitivas empleadas en el diseño de aplicaciones orientadas a objetos enriquecidas con relaciones entre objetos. Este modelo de desarrollo se basa en el empleo de tres frameworks: class framework, composition framework y GUI framework.

El class framework consiste de una biblioteca de clases reutilizable. Para realizar la identificación de las clases necesarias de la aplicación se utilizan las técnicas estándar de diseño orientado a objetos. El composition framework consiste de una biblioteca de clases link. Esta biblioteca permite que los usuarios utilicen las clases (links) predefinidas y las extiendan vía mecanismo de herencia. Entre las clases link básicas se encuentran: SimpleLink, NavigationalLink, NodeToNode, SpanToNode, StructureLink, SetLink, ListLink. El último paso de esta metodología consiste en el diseño de la interfaz gráfica del usuario empleando elementos del GUI framework. En esta fase deben quedar determinadas las ventanas, el tipo de presentación que será mostrada en cada ventana, la forma en que serán presentados los atributos, las funcionalidades de cada objeto visual, etc.

Para desarrollar aplicaciones hypermedia usando este método puede emplearse ONTOS Studio. Esta herramienta case provee una interfaz gráfica que genera código C++ que implementa el modelo hypermedia especificado.

4.4 OOHDM

Object Oriented Hypermedia Method considera al desarrollo de una aplicación hypermedia como un proceso basado en cuatro fases: diseño del esquema conceptual, diseño del esquema navegacional, diseño de la interfaz abstracta e implementación.

El resultado de la primer etapa es el diseño de un modelo orientado a objetos del dominio de la aplicación. Para su especificación se utilizan las primitivas de diseño orientado a objetos estándar con el agregado de relaciones y atributos. Como sucede en cualquier diseño orientado a objetos, los mecanismos de especificación empleados en esta etapa son clases conceptuales basadas en agregación y jerarquías de clases.
En el diseño del esquema navegacional se especifican, de acuerdo a los tipos de usuario que tendrá la aplicación, caminos sobre el esquema conceptual. Los artefactos empleados en este esquema son contextos navegacionales, nodos, aristas y aspectos dinámicos de navegación.
El diseño de la interfaz del usuario se define de manera abstracta. En esta etapa se especifican las interacciones entre los objetos de la interfaz, los objetos del dominio que serán visualizados y las transformaciones que pueda realizar la interfaz.
Finalmente, en la etapa de implementación, se realiza el mapeo de los objetos de los esquemas previos a objetos concretos. Existen diversas plataformas de implementación y herramientas case que facilitan el desarrollo de una especificación OOHDM. Entre las plataformas se encuentran Hypercard [Com89], Toolbook y Microcosm [HDH96] y entre las herramientas se encuentran WCML [GSG99].

	Fases de Diseño	Técnica de Diseño	Herramientas
HDM		Entidad Relación	
RMM	E-R design Slice design Navigational design Conversion protocol design User interface design Run-time behavior design Construction and testing	Entidad Relación + Slice + RMDM diagrams	RM-Case
EORM	Class framework Composition framework GUI framework	Orientado a Objetos	ONTOS Studio
OOHDM	Conceptual schema design Navigational schema design Abstract interface design Implementation	Orientado a Objetos + relaciones + atributos + ADVs + Statecharts	OOHDM-Web

Figura 4.1: *Características de HDM, RMM, EORM y OOHDM.*

La figura 4.1 presenta un cuadro que resume las fases de diseño, las técnicas empleadas y las herramientas disponibles para desarrollar aplicaciones con cada una de las metodologías mencionadas. Para el desarrollo de este trabajo nos inclinamos por el empleo de OOHDM porque provee la definición de un modelo de la aplicación desacoplado del esquema navegacional y este de la interfaz del usuario posibilitando así realizar un mantenimiento sencillo y una fácil comprensión y análisis de la aplicación. Además, su perfil orientado a objetos brinda facilidad para realizar cualquier tipo de extensión a la aplicación.

Capítulo 5

Introducción a OOHDM

OOHDM (Object Oriented Hypermedia Design Method) [SR95b] es un método de ingeniería de software que considera al desarrollo de una aplicación hypermedia como un proceso de cuatro fases: diseño del esquema conceptual, diseño del esquema navegacional, diseño de la interfaz abstracta del usuario e implementación. A continuación se describe sintéticamente cada una de estas fases.

5.1 Esquema Conceptual

El desarrollo del esquema conceptual de la aplicación es la primera etapa de la metodología. Esta fase consiste en realizar el diseño de un modelo del dominio de la aplicación utilizando los principios de diseño de aplicaciones orientadas a objetos [RBP+91] con el agregado de algunas primitivas tales como atributos y subsistemas.

En esta etapa, se definen clases conceptuales utilizando mecanismos de agregación y jerarquías de generalización/especialización sin tener en cuenta los tipos de usuarios que tendrá la aplicación ni sus funcionalidades. Los objetos y clases definidos en el esquema se unen entre sí mediante relaciones. Dado que los objetos son instancias de clases, el empleo de relaciones entre clases abstrae la relación entre objetos. La definición de subsistemas se basa en abstracciones de sistemas complejos. Cada uno de estos puede tener uno o varios puntos de entrada y, al igual que las relaciones entre clases, también pueden existir relaciones entre clases y subsistemas. Los atributos representan propiedades intrínsecas o conceptuales de los objetos, pueden ser de clases o de relaciones y además tipados. El tipo del atributo representa una relación implícita, una apariencia visual o una apariencia retórica.

5.2 Esquema Navegacional

La navegación es una las principales características de una aplicación web y por tal motivo OOHDM considera a esta fase como la etapa crucial. En el transcurso de esta etapa se definen los objetos navegacionales que la componen y se especifican los diferentes caminos que pueden recorrer el esquema conceptual. La definición de estos caminos es importante porque de esta manera se mapea una forma diferente de recorrer el espacio de información. Dado que estos caminos se obtienen a partir del mismo esquema conceptual, se dice que cada uno de ellos actúa como un observador del dominio.

Al realizar el diseño del esquema que corresponde a esta etapa se consideran los siguientes aspectos: los objetos que podrán ser navegados, la relación existente entre los objetos navegables y los del esquema conceptual, la estructura subyacente de navegación, los contextos por los cuales el usuario navegará, la apariencia que tendrán los objetos navegables y los efectos que pueden aparecer al navegar de un objeto a otro. Todos estos aspectos quedan definidos al especificar contextos navegacionales, nodos, aristas y aspectos dinámicos de la navegación.

Los contextos navegacionales describen la estructura navegacional general de la aplicación hypermedia [SR95a]. Cada contexto está compuesto por un conjunto de nodos, aristas y otros contextos navegacionales anidados. Sus funciones principales son auxiliar a la organización de los objetos navegables y establecer espacios navegacionales consistentes. Ambas funcionalidades ayudan a disminuir las posibilidades de que el usuario pierda la orientación dentro del hiperespacio.

Los nodos y las aristas especifican los objetos que serán visibles por los usuarios. Los nodos representan "ventanas"lógicas sobre las clases definidas en el esquema conceptual mientras que las aristas son derivaciones de las relaciones conceptuales. Los nodos pueden ser atómicos o compuestos, son descritos a través de atributos y anclas en el caso de ser atómicos y como un conjunto de nodos componentes en el caso de ser compuestos. Al ser inmersos dentro de los contextos navegacionales, se define un conjunto de clases decoradoras que adaptan cada nodo en el contexto necesario. Las aristas relacionan objetos navegables y establecen relaciones de cardinalidad uno a uno o uno a muchos. El resultado de la travesía de una arista se expresa mediante la definición de una semántica navegacional como comportamiento de la arista. Otras estructuras de acceso, como índices, también son definidas como clases y se presentan como nodos alternativos para navegar la aplicación.

Si bien los nodos, las aristas y los contextos navegacionales definen una estructura estática dentro de la aplicación hypermedia, en este esquema también se especifican aspectos dinámicos de la navegación. Para tal objetivo se emplean Diagramas de Navegación. Este modelo se basa en la definición de máquinas de estado [HPSS87] en el cual se muestran las transformaciones suscitadas en el estado del espacio navegacional.

5.3 Interfaz Abstracta

Durante el diseño de la interfaz abstracta, se define el modelo de la interfaz del usuario. En esta etapa se especifican las interacciones entre los objetos de la interfaz, los objetos del dominio que serán visualizados y las transformaciones que pueda realizar la interfaz. En términos más concretos, se especifican los siguientes aspectos: la apariencia de cada objeto navegable que será percibido por el usuario (figura, texto, etc.), los objetos propios de la interfaz (botones, menús, etc.), las relaciones entre los objetos de la interfaz y los objetos navegables, las transformaciones que suceden por el efecto de la navegación o de eventos externos y la sincronización entre los objetos que forman la interfaz.

Las entidades empleadas para la especificación de los aspectos mencionados se llaman ADV (Abstract Data View). Tales entidades son objetos abstractos que poseen un estado interno y un protocolo de comunicación sin implementación [CMHCL94] [CILS93] [CL95]. Un ADV incluye un conjunto de atributos y un conjunto de eventos. Cada atributo define un estilo específico como por ejemplo su posición dentro del documento, color o sonido y cada evento especifica un canal de comunicación para "recibir"los eventos generados por los usuarios. En la especificación de la interfaz, un ADV puede ser compuesto por otros ADVs mediante relaciones de agregación o composición, lo cual permite especificar una interfaz usando objetos anidados. Un ADV también puede ser subclasificado en clases de generalización/especialización proveyendo así una estructura poderosa para definir jerarquías de objetos de interfaz.

5.4 Implementación

En la etapa de implementación se realiza el mapeo de los objetos de los esquemas anteriores a objetos concretos de un lenguaje. Existen diversas plataformas de implementación y herramientas case que facilitan el desarrollo. Entre las plataformas se encuentran Hypercard, Toolbook y Microcosm y entre las herramientas se encuentran WCML que provee un framework de componentes XML y una

compilador que al interpretar las componentes generan código en HTML.

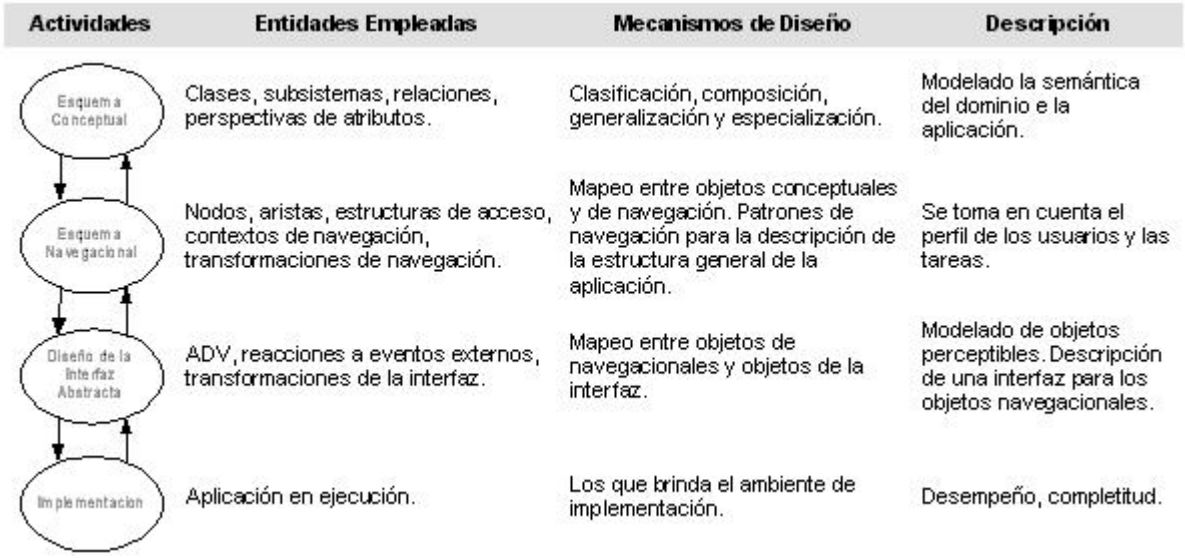

Figura 5.1: *Las cuatro fases de OOHDM.*

La figura 5.1 muestra un cuadro que sintetiza las entidades empleadas en cada fase, los mecanismos de diseño que se utilizan y la descripción de cada actividad.

Capítulo 6

El Tiempo en la Interfaz del Usuario

La interfaz del usuario de una aplicación hypermedia se compone, entre otras cosas, de entidades o elementos multimedia encargados de mostrar, de acuerdo a su tipo, la información contenida en el sistema. Estas entidades tienen como soporte de contención los documentos de la aplicación y son presentadas al usuario a medida que recorre los contextos navegacionales de la aplicación.

La manera en que se realiza la presentación de las entidades multimedia puede ser interpretada de varias formas. Una de ellas es suponer que el documento posee un comportamiento que administra su propia presentación y la de todas sus componentes. Otra interpretación de lo mismo es suponer que el documento delega a otra componente de software la responsabilidad de administrar todas acciones de visualización. Una interpretación más consistente es suponer que existe un grupo de componentes que interactúan entre sí para coordinar la presentación de las entidades multimedia que conforman el documento.

El primer enfoque proyecta al documento como si fuera una entidad inteligente capaz de contener y administrar cualquier tipo de presentaciones. El segundo lo muestra como una entidad de responsabilidades limitadas que colabora con un scheduler o administrador de entidades multimedia. Mientras que el tercer enfoque es más robusto porque agrega contenido y además desacopla la responsabilidad de realizar la presentación hacia otras componentes. De todas, la tercer interpretación es la más apropiada porque da forma a una arquitectura de componentes robusta cuyo objetivo es realizar la presentación de las entidades del documento. La figura 6.1 muestra las tres interpretaciones.

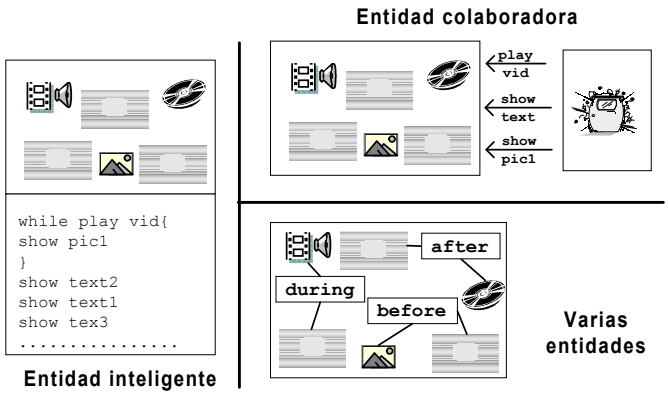

Figura 6.1: *Tres formas de interpretar la visualización de entidades multimedia.*

En base al último enfoque, en la siguiente sección se analiza cual es el modelo teórico apropiado para especificar, en la etapa de diseño, los aspectos temporales de la interfaz del usuario. A continuación se analizarán distintos tipos de escenarios temporales que pueden suceder entre las entidades multimedia de un documento hypermedia. En éstos se consideran las interacciones que se llevan a cabo entre entidades que comparten la misma posición y las que tienen lugar entre entidades multimedia localizadas en diferentes posiciones. Luego se propondrá una arquitectura de software orientado a objetos para especificar los aspectos temporales de la interfaz del usuario en la etapa de diseño de una aplicación hypermedia.

6.1 Elección del Modelo de Tiempo

Los modelos de especificación de tiempo más empleados en la etapa de diseño de las aplicaciones multimedia son los modelos point-based y los modelos interval-based. En los primeros la unidad elemental de especificación es un punto del espacio de tiempo que refleja cada evento que sucede en la presentación. En los modelos interval-based las entidades elementales de especificación son intervalos que se relacionan mediante las relaciones mostradas en la figura 6.2.

Algunos trabajos utilizan un punto de vista post implementación al momento de realizar una mirada crítica sobre los modelos de especificación de aspectos temporales. En particular, en [DK96] se menciona que el empleo de las relaciones de intervalos de tiempo son un inconveniente porque no ayudan a especificar relaciones entre intervalos que poseen duraciones de tiempo desconocidas.

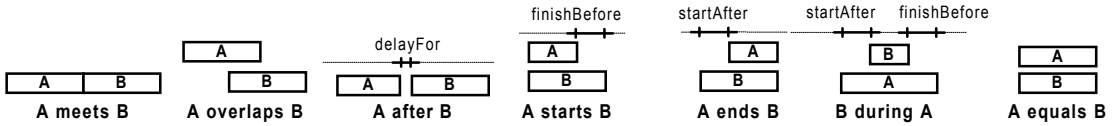

Figura 6.2: *Relaciones de intervalos de tiempo.*

Además, también se hace referencia al carácter poco descriptivo de las relaciones como otro gran problema. Esto último se debe a que las relaciones pueden expresar una configuración estática de intervalos pero no una relación casual o funcional entre los mismos.

Nosotros sostenemos que los aspectos temporales deben ser especificados en la etapa de diseño y no interpretados luego de la implementación de la aplicación. Por tal motivo, desde el punto de vista del diseño, opinamos que las relaciones de intervalos de Allen son más que apropiadas para especificar escenarios temporales en la etapa de diseño de la aplicación y que además describen tres características muy importantes: *de que manera* se ejecutan los intervalos, *por cuanto tiempo* los intervalos son ejecutados y *en que momento* cada intervalo comienza (o finaliza) su ejecución durante la relación. La primera y tercera característica están fuertemente relacionadas y pueden ser vistas como un mecanismo atómico de sincronización. Por ejemplo, la relación MEETS especifica que el intervalo B se ejecuta inmediatamente a continuación del A (sin tiempo de demora entre la presentación de ambos). La segunda característica puede ser considerada como un atributo propio del intervalo porque solamente especifica el tiempo durante el cual se realiza una acción. A lo largo del trabajo nos referiremos a ese mecanismo de sincronización con el término *synchronization constraint*.

En base a lo expuesto anteriormente, proponemos usar el modelo interval-based y, para agregar flexibilidad al diseño, emplear un enfoque orientado a objetos. Esto nos permitirá desacoplar las tres características anteriormente mencionadas y encapsularlas en clases, objetos, atributos, mensajes y jerarquías de clases.

6.2 Intra e Inter Frame Synchronization

Durante la presentación de un documento hypermedia se puede advertir la existencia de algunos aspectos temporales inherentes a la presentación. Dos de estos aspectos son la manera y el orden en que ocurren las presentaciones de los elementos multimedia. Dado que es importante que estas características sean especificadas en el transcurso de la etapa de diseño, en esta sección se estudian

dos escenarios de presentación de los documentos hypermedia. El primero de ellos analiza el caso en que los elementos multimedia comparten la misma posición del documento. Mientras que el segundo comprende el caso en que los elementos multimedia están situados en diferentes posiciones del mismo documento. Llamaremos *Intra Frame Synchronization* al primer escenario e *Inter Frame Synchronization* al segundo.

Intra Frame Synchronization

En la actualidad, las páginas web son la clase de documentos más explotada en el desarrollo de aplicaciones hypermedia. Es habitual encontrar entre sus contenidos imágenes que se visualizan de forma intercalada sobre una misma posición del documento. Tal efecto es fácil de lograr empleando una entidad multimedia llamada banner. Este elemento se compone de una lista de imágenes y un mecanismo interno que permite que todos los elementos de su lista sean presentados, sobre una misma posición, uno a término del otro y en secuencia. La figura 6.3 muestra la página web de una compañía cinematográfica que utiliza un banner para publicitar estrenos de películas.

Figura 6.3: *Banners en la interfaz del usuario de una aplicación hypermedia.*

Supongamos por un momento que es posible programar un banner con contenido audible. Para tal objetivo se necesitaría incluir un tiempo de demora después de la reproducción de cada fragmento de audio de manera tal que el usuario de la aplicación pueda ser capaz de comprender la información transmitida. De esta forma, podemos observar que el banner no solo presenta su contenido de manera secuencial sino que además distribuye todo el tiempo de presentación entre sus componentes.

De éste análisis se puede interpretar que el comportamiento de un banner involucra tres partes principales: el orden de presentación de sus elementos, el tiempo de presentación de cada elemento y el tiempo de demora entre las presentaciones. Dado que la presentación de sus elementos sucede una a continuación de la otra, se puede asegurar que el orden de presentación de todos estos es secuencial. Si bien el tiempo de presentación es propio de cada elemento, podría darse el caso en que el banner restrinja ese valor a uno mucho más pequeño. Con lo cual, en el mejor de los casos, el tiempo de presentación de cada componente coincidiría con el asignado por el banner. Tal como se analizó anteriormente, el tiempo de demora entre las presentaciones puede tomar diferentes valores de acuerdo al origen del medio incorporado. Por lo tanto, esta última componente es tan o más importante de considerar que las anteriores. A continuación, la figura 6.4 muestra en detalle las componentes recientemente descritas.

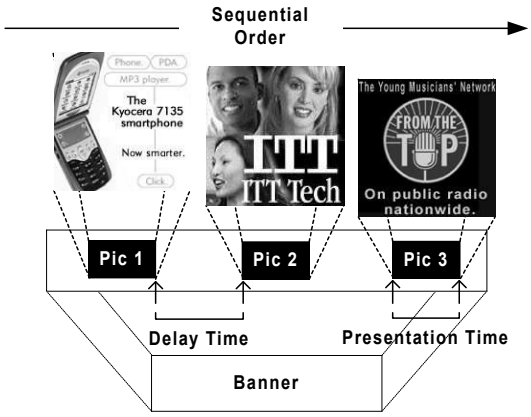

Figura 6.4: *Composición de un banner.*

Una manera de interpretar éste comportamiento es emplear un esquema de sincronización secuencial. Este es el mismo que emplea el modelo de algunas de relaciones de intervalos de tiempo. Por ejemplo, si un banner incluye un tiempo de demora entre las presentaciones de sus componentes, entonces su comportamiento puede modelarse utilizando una relación AFTER. De otro modo, si tal tiempo no esta incluido o es nulo, entonces su comportamiento se puede reflejar utilizando una relación MEETS. De una u otra manera, para modelar ambos conceptos, es necesario el empleo de algún tipo de mecanismo de control temporal. Por lo tanto, haremos empleo de synchronization constraints definidas en la sección anterior.

Al seleccionar las synchronization constraints, es necesario asegurar que éstas permitan mapear por completo las componentes anteriormente descritas. Esto significa que deben proveer un orden de ejecución secuencial y además permitir asignar eventualmente un tiempo de demora entre sus componentes.

Usando un modelo de diseño orientado a objetos, ambos tipos de restricciones pueden ser mapeadas con una jerarquía de clases. Esta metodología de diseño permite desacoplar los aspectos generales de los particulares. Así, la clase abstracta se encargará de componer los elementos multimedia a coordinar y proveer un orden secuencial entre sus presentaciones. Por otro lado, el comportamiento relacionado al tiempo de demora entre las presentaciones quedará como responsabilidad de las subclases. La figura 6.5 muestra la jerarquía de clases.

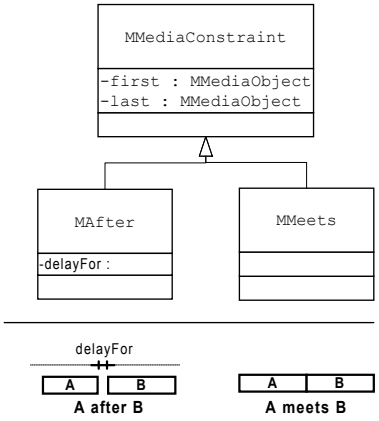

Figura 6.5: *Jerarquía de clases para modelar sincronización secuencial.*

En la figura se pude observar que en la clase abstracta `MMediaConstraint` las variables de instancia `first` y `last` permiten componer los elementos multimedia a presentar. Sobre la misma entidad, el orden secuencial de presentación se logra mediante una interfaz abstracta a implementar con librerías de algún lenguaje de programación específico. En el nivel concreto, la subclase `MAfter` y `MMeets` mapean las relaciones AFTER y MEETS respectivamente. Sus estructuras difieren una de la otra en el sentido que la primera permite al diseñador definir un tiempo de demora entre dos presentaciones. Asimismo, éste atributo también marca la diferencia de comportamiento entre ambas subclases.

En la práctica, un banner es un programa que se incluye dentro de código HTML como cualquier figura de tipo JPG o GIF. Una manera diferente de proveer las mismas características que tiene

un banner es utilizando un programa java Applet. Este tipo de programas son mucho más que un simple banner. Entre otras funcionalidades, permiten reproducir sonido y video. Cualquiera que sea, ambos pueden ser interpretados de forma general. En este trabajo, estos tipos de programa serán tratados como instancias de la clase `MMediaFrame` y cualquier elemento multimedia que forme parte del contenido de estos objetos serán vistos como una instancia de la clase `MMediaObject`. Así, para completar este primer modelo de especificación temporal es necesario combinar ambas clases de objetos con la jerarquía anteriormente definida. La figura 6.6 muestra la arquitectura resultante.

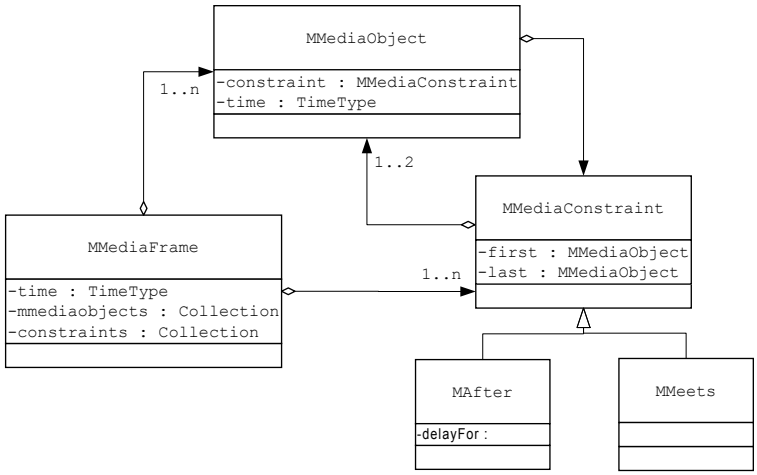

Figura 6.6: *Arquitectura para la especificación de aspectos temporales entre entidades multimedia que comparten la misma posición en la interfaz del usuario.*

De esta manera, el concepto de media element time [HvOM+99] toma forma de atributo en la clase `MMediaFrame`. Su valor es el resultado de la suma del media element time de cada `MMediaObject` y el valor del tiempo de demora especificado en cada synchronization constraint.

Inter Frame Synchronization

En la sección anterior se analizó un escenario en el que los elementos multimedia comparten la misma posición dentro de la interfaz del usuario . En esta sección se supondrá que éstos están situados en diferentes posiciones. Dentro de este escenario la presentación de cada entidad puede suceder antes, después o durante la presentación de otra entidad. Por ejemplo, si se consideran una interfaz compuesta por un video y una imagen, entonces puede suceder que el video se proyecte antes de la imagen o bien que ambos se presenten simultáneamente. La figura 6.7 muestra un ejemplo.

Figura 6.7: *Presentación consecutiva y simultánea de dos entidades multimedia.*

Una manera de interpretar ambos comportamientos es empleando un esquema de sincronización secuencial y un esquema de sincronización paralelo. Ambos se disponen en el modelo de intervalos de tiempo de Allen. Por tanto, si la presentación de un recurso multimedia debe suceder a continuación del otro, entonces se puede emplear un esquema de tipo secuencial. De otra forma, si la presentación de un recurso multimedia debe suceder durante, al comienzo o al finalizar otra presentación, entonces se puede emplear un esquema de tipo paralelo. De la misma manera que en la sección anterior, para modelar ambos conceptos se hará empleo de synchronization constraints.

De esta manera, usando un modelo de diseño orientado a objetos, proponemos definir las jerarquías de clase `SeqConstraint` y `ParConstraint` para modelar la ejecución de eventos en orden secuencial y paralelo respectivamente. En la figura 6.8 se puede apreciar que la jerarquía de clase `SeqConstraint` modela las relaciones de intervalos `Meets` y `After`. Mientras que las relaciones `Starts`, `Finishes`, `During`, `Overlaps` e `Equals` quedan modeladas con la jerarquía `ParConstraint`. Además, la figura también muestra la correspondencia existente entre el modelo de intervalos de tiempo de Allen y las subclases de cada jerarquía.

El empleo de jerarquías de clases tiene dos razones fundamentales. La primer razón es que el mecanismo de presentación secuencial y paralelo queda encapsulado en las clases abstractas. Esto permite generalizar la manera en que se realiza la presentación en un nivel de especificación más alto. La segunda razón es que el momento en que se realiza cada tipo de presentación quedan modelado por las clases concretas. Por esto mismo la clase `After` es capaz de especificar un tiempo de demora

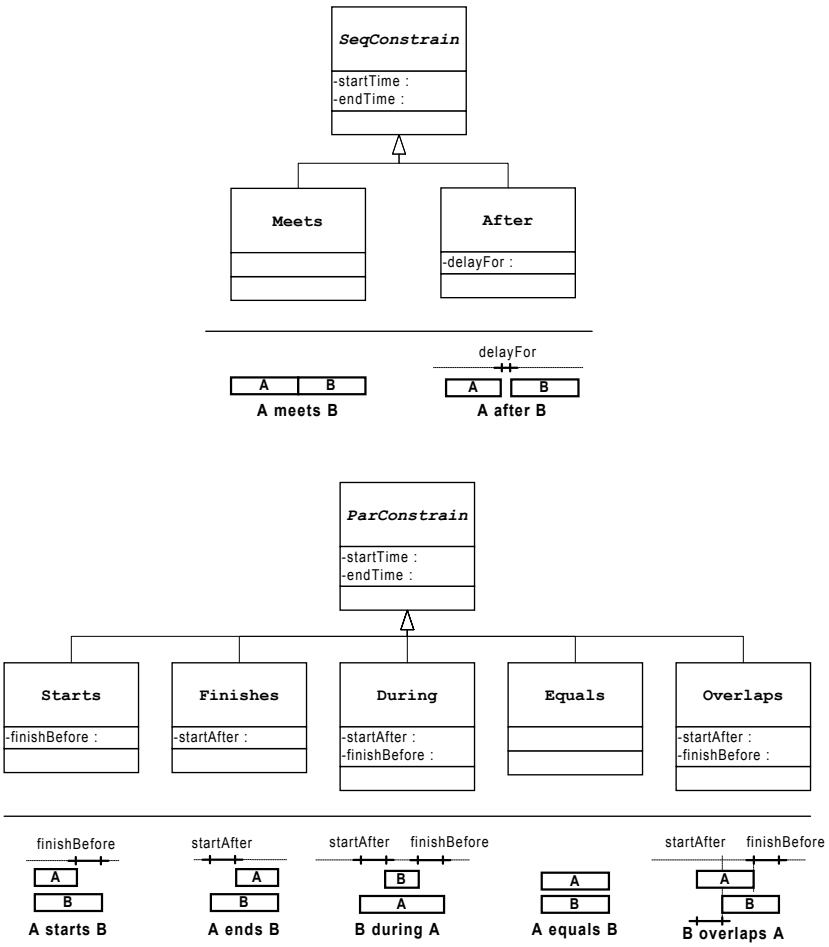

Figura 6.8: *Jerarquía de clases para modelar sincronización secuencial y paralela.*

entre dos presentaciones simplemente agregando la variable de instancia `delayFor` mientras que la clase hermana `Meets` no la necesita para denotar que una presentación sucederá inmediatamente después de finalizada la otra. De la misma forma, en la otra jerarquía de clases, se puede observar que la clase `Starts` usa la variable de instancia `finishBefore` para representar el momento en el cual la primer presentación deja de estar disponible. El mismo concepto pero opuesto es mantenido por la variable de instancia `startAfter` de la clase `Finishes`. El concepto de ambas variables de instancia se combina dentro de la clase `During` y `Overlaps` en tanto que `Equals` indica que dos presentaciones estarán disponibles durante exactamente el mismo tiempo.

Durante la etapa de diseño de una aplicación, el autor puede necesitar especificar sincronización entre dos, tres o más recursos multimedia. Por lo tanto, usaremos el término *simple synchronization* para la sincronización entre dos elementos multimedia y el término *composite synchronization* para la sincronización entre tres o más de ellos. La figura 6.9 muestra un ejemplo gráfico de ambas alternativas.

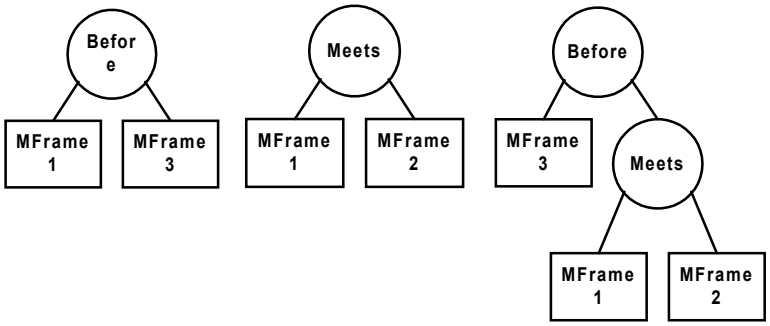

Figura 6.9: *Sincronización simple y compuesta entre entidades multimedia.*

En la figura se puede observar que la sincronización de tipo simple synchronization emplea un synchronization constraint entre dos `MMediaFrame`. Mientras que la de tipo composite synchronization emplea un synchronization constraint entre un `MMediaFrame` y otro synchronization constraint. Para modelar ambos conceptos proponemos la definición de una jerarquía de clases que adapta los objetos a ser sincronizados con el synchronization constraint elegido. La figura 6.10 muestra tal jerarquía de clases.

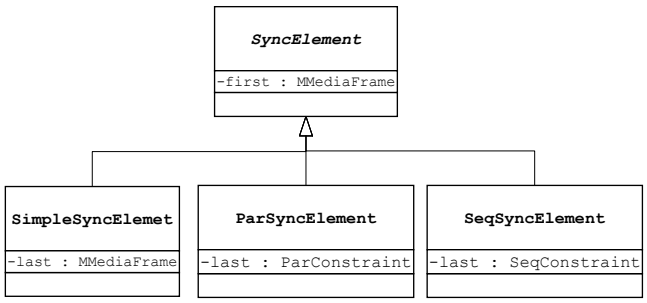

Figura 6.10: *Jerarquía de clases para componer sincronización entre dos o más entidades.*

De esta manera, la clase abstracta `SyncElement` abstrae el tipo de cliente que tendrá el synchronization constraint a ser empleado. La clase concreta `SimpleSyncElement` modela un cliente compuesto por dos `MediaFrame` y las clases concretas `ParSyncElement` y `SeqSyncElement` modelan clientes compuestos por un `MediaFrame` y otro synchronization constraint. La definición de una clase para cada tipo de synchronization constraint tiene como objetivo desacoplar el tipo de comportamiento que pueda darse entre las entidades multimedia a sincronizar.

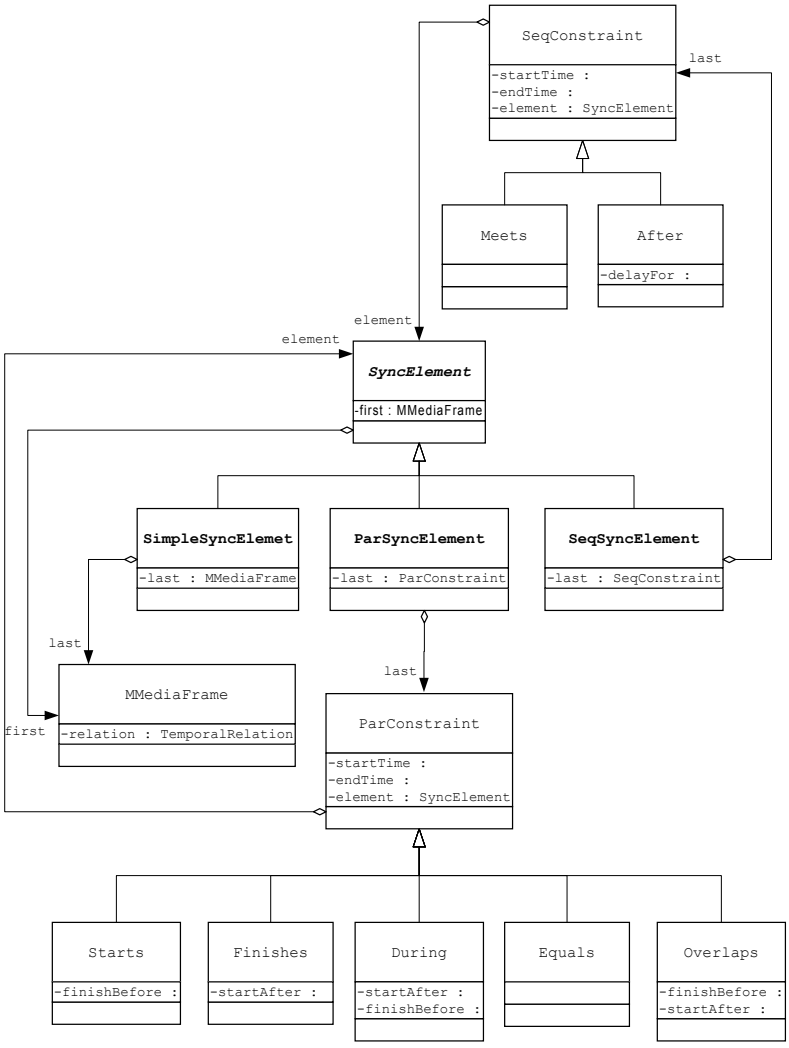

Figura 6.11: *Arquitectura para la especificación de aspectos temporales entre entidades multimedia que no comparten la misma posición en la interfaz del usuario.*

Finalmente, para disponer una arquitectura orientada a objetos apropiada para especificar aspectos temporales entre los recursos multimedia de un documento, se deben componer las tres jerarquías de clases ya definidas como lo muestra la figura 6.11. En ella se puede observar que el concepto media element time [HvOM$^+$99] es una propiedad del objeto `MultimediaFrame`. De la misma forma que la sección anterior, su valor es el resultado de la suma del media element time de cada `MMediaObject` mas el valor del tiempo de demora eventual que está especificado en cada synchronization constraint.

Capítulo 7

El Tiempo en la Estructura Navegacional

En la actualidad existen diferentes enfoques que definen hypermedia. Por un lado están aquellos que lo definen como la combinación de hypertext y multimedia. Desde este punto de vista, el lado hypertextual de la expresión describe la forma que tiene la estructura o el modelo lógico de la aplicación. Mientras que el perfil multimedial enmarca los tipos de dispositivos de presentación empleados, es decir videos, imágenes, texto o sonido. Por otro lado, se encuentran aquellos enfoques que definen hypemedia como la presentación de la información mediante el empleo de diferentes tipos de medios y asistida por la navegación. Si bien este enfoque utiliza el mismo perfil multimedial que el anterior, es más general porque no hace referencia alguna a la forma de la estructura de la aplicación. Por ejemplo, en la primer definición, el término hypertext esta sujeto al empleo de hyperdocuments y anclas mientras que la segunda definición se admite el empleo de un modelo más elaborado. Es decir, que la estructura subyacente puede ser una base de datos o bien una aplicación orientada a objetos como lo muestra la figura 7.1.

Al involucrar el término navegación, la segunda definición agrega una característica diferente: un mismo mapa o modelo navegacional puede no incluir todas las entidades del modelo lógico. En efecto, en una aplicación pueden existir entidades que solamente son accedidas por grupos de usuarios. En otras palabras, el modelo de navegación puede no ser idéntico al modelo lógico. Esta característica es muy importante porque, en la etapa de diseño, nos permite desacoplar la estructura de la aplicación respecto de la manera en que se recorren las entidades de la aplicación. En algunas metodologías

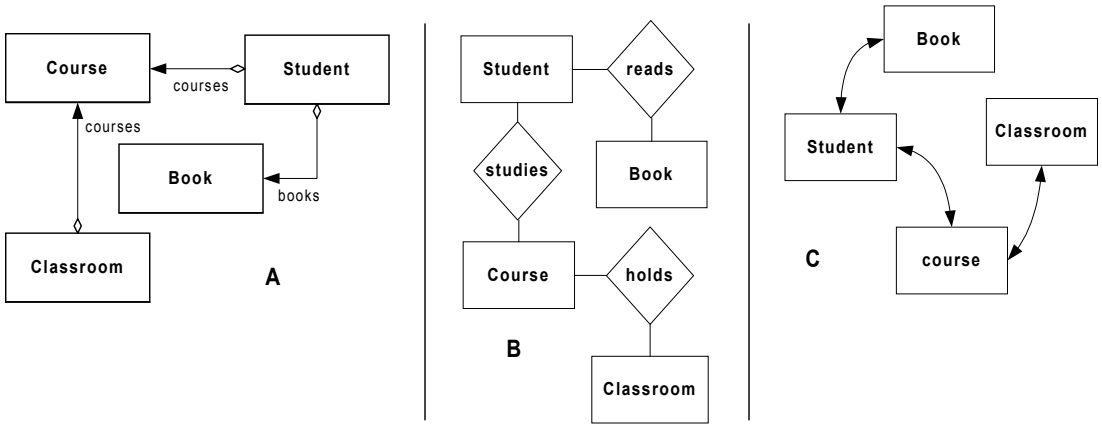

Figura 7.1: *Tres maneras de organizar la información: modelo orientado a objetos (A), modelo entidad relación (B), modelo de hyperdocuments (C).*

de diseño, tomar en cuenta esta característica hace posible que el diseñador de la aplicación pueda definir diferentes tipos de vistas o contextos de navegación para la misma aplicación [SR95b].

7.1 Identificación del Aspecto Temporal

A lo largo de la vida útil de una aplicación hypermedia, su modelo lógico y su modelo navegacional pueden ser modificados de acuerdo requerimientos de diferente naturaleza.

La mayoría de las modificaciones que ocurren en la estructura de la aplicación están relacionadas a la expansión o crecimiento del sistema. Por ejemplo una posible modificación en la estructura de una aplicación que modela el funcionamiento de una universidad es la incorporación de componentes de software para modelar la realización de un congreso. Otro ejemplo del mismo tipo son las aplicaciones que modelan el data mining de una empresa. En estos casos es muy común la incorporación de pequeñas aplicaciones que proveen nuevas funcionalidades como por ejemplo la entrega de productos a clientes.

Dentro de las modificaciones que ocurren en el modelo navegacional una gran variedad están ligadas a un aspecto temporal. Los sitios web desarrollados para el envío de publicaciones a congresos y los e-commerce son dos de los mejores ejemplos ilustrativos de este tipo de modificación.

En los sitios de envío de publicaciones, es muy común que el modelo navegacional se componga de una o varias entidades que den acceso a las herramientas para enviar trabajos. En la figura 7.2 se muestran los dos escenarios posibles de este ejemplo. En el primer escenario se ve que el acceso a las herramientas es viable siempre y cuando la fecha límite para el envío de trabajos aún no haya

transcurrido. En caso que esto suceda, como se muestra en el segundo escenario, las herramientas ya no serán accesibles.

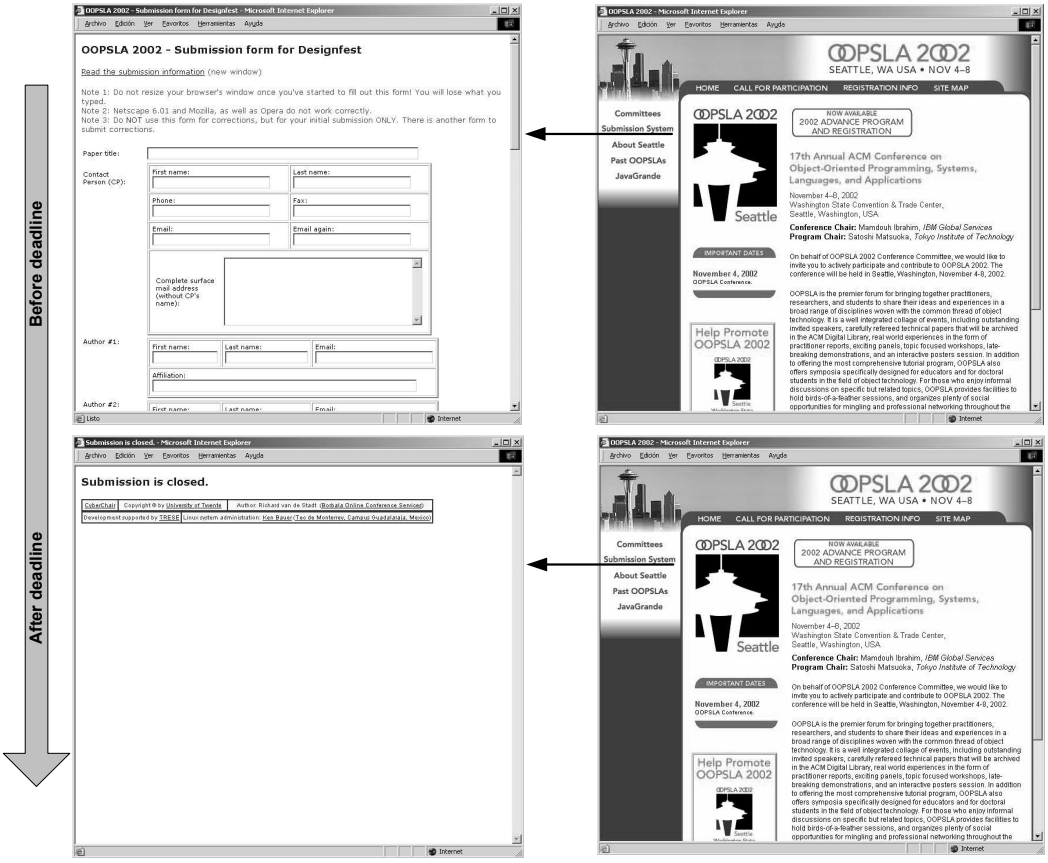

Figura 7.2: *Páginas usadas para el envío de publicaciones.*

Un comportamiento similar sucede en las aplicaciones e-commerce. En este sentido, es habitual encontrar sitios donde la oferta de un grupo de productos permanece disponible durante un determinado tiempo. Inclusive, en algunas aplicaciones comerciales, este escenario sucede de manera iterativa como es el caso de aquellos en donde las ofertas están sujetas a una temporada que se repite. Así, en la figura 7.3 se puede observar que la posibilidad de acceder a los productos a bajo precio esta sujeta al tiempo que dure la temporada de verano. Luego, una vez que tal temporada concluya, los productos a bajo precio ya no podrán ser accedidos.

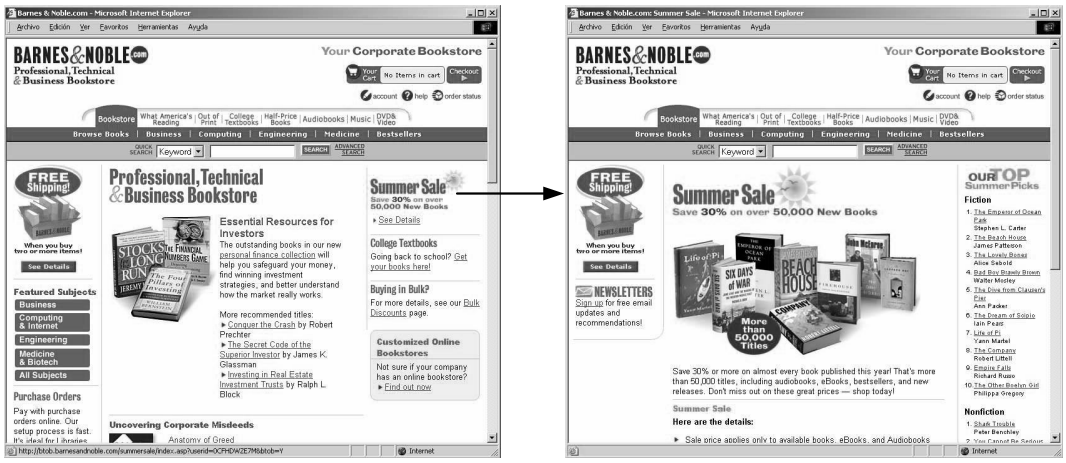

Figura 7.3: *Características temporales de una aplicación e-commerce.*

En los ejemplos recientemente descritos, se observa claramente la aparición de una componente temporal que modifica el modelo navegacional de la aplicación. En este trabajo consideramos que la especificación de esta componente en la fase de diseño es importante. Por tal motivo, en el siguiente capítulo se propone una forma de especificar esta componente de tiempo usando un modelo orientado a objetos.

Capítulo 8

Modelos de Tiempo para la Estructura Navegacional

El objetivo de esta sección es definir una arquitectura que permita especificar el aspecto temporal en la etapa de diseño del modelo navegacional de la aplicación. Para cumplir con este objetivo, primero serán identificadas las entidades más importantes que la componen. Luego se propondrán los posibles escenarios de interacción entre las componentes y, finalmente, se dará a conocer una arquitectura de objetos que permita definir y modelar el aspecto temporal de una aplicación.

8.1 Identificación de las Componentes

A partir de los ejemplos citados en el capítulo anterior, se puede observar que la página origen, la página destino y el link son las entidades básicas que componen esta arquitectura de tiempo. A ellas se le suma una componente extra que define el momento preciso en el cual el link deja de relacionar ambas páginas. Las cuatro componentes se detallan en los modelos navegacionales reducidos de la figura 8.1.

Desde un punto de vista más abstracto, se puede decir que la arquitectura esta formada por dos componentes cualesquiera del modelo de la aplicación, una tercer componente que las relaciona y una cuarta que especifica dos atributos importantes: un momento en el tiempo y un comportamiento especifico asociado a ese momento.

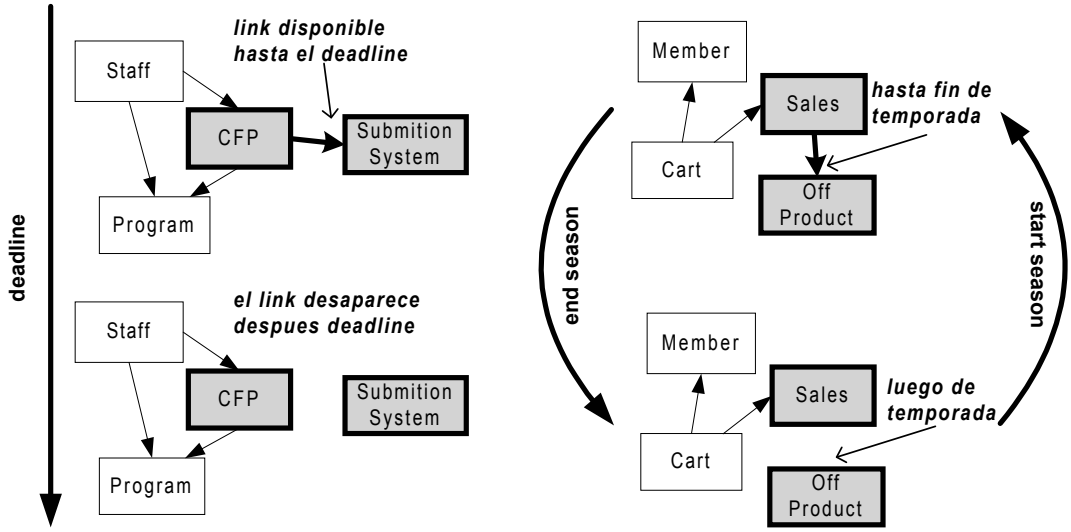

Figura 8.1: *Componentes del aspecto temporal.*

8.2 Tipos de Interacción

De acuerdo a las componentes anteriormente identificadas, la especificación del aspecto temporal queda modelada mediante una componente que asocia un momento en el tiempo y una acción a ejecutarse. Volviendo sobre la figura 8.1, en el primer modelo navegacional, la acción que se realiza como consecuencia del deadline hace que desaparezca el link que relaciona la componente `CFP` con la componente `SubmitionSystem`. De la misma manera, en el segundo modelo navegacional, una vez que alguna de las dos temporadas transcurre se ejecuta una acción que hace desaparecer o reaparecer el link entre las entidades `Sales` y `OffProduct`.

Existen diferentes maneras de interpretar el comportamiento que emerge de estas cuatro componentes. En primer lugar podemos suponer que el link desaparece del modelo navegacional. Dado que es muy utópico imaginar que un link desaparece cuando una aplicación esta en producción, proponemos definir una acción cuya responsabilidad es validar si corresponde mantener activo el link. De esta forma, cuando un usuario recorra desde una entidad hacia otra, esta componente decidirá sobre la validez del camino entre a las dos componentes.

Un enfoque diferente es suponer que la entidad destino es reemplazada por una nueva entidad. Para que este enfoque se lleve a cabo proponemos la definición de una acción que ejecute un algoritmo capaz de decidir como se asocian dos entidades en un instante de tiempo. En otras palabras, cuando el usuario recorra desde una entidad a otra, esta componente ejecutará un algoritmo que definirá cual es el verdadero destino del link. La figura 8.2 ilustra ambas interpretaciones.

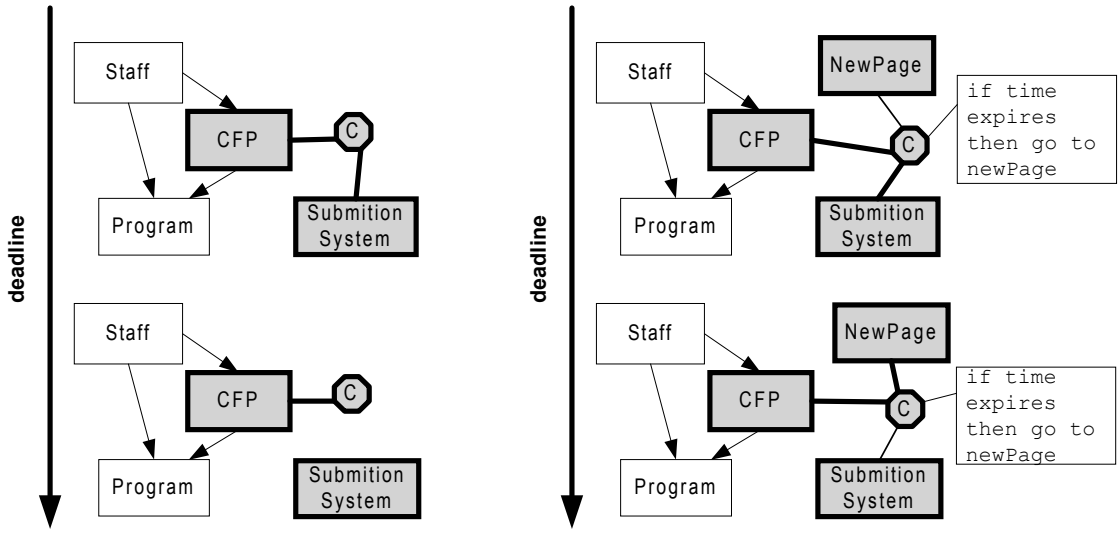

Figura 8.2: *Interacción entre componentes del aspecto temporal.*

A partir del análisis de estas componentes y de sus posibles interacciones, se pueden definir las clases necesarias para especificar el aspecto temporal en la etapa de diseño. A continuación se describe una arquitectura que incluye la definición y composición de las componentes descritas.

8.3 Arquitectura de Componentes

En la sección anterior se identificaron y describieron las componentes principales que están presentes en el momento que ocurre la modificación del modelo navegacional de una aplicación hypermedia. Dado que el objetivo de este trabajo es proponer un modelo de diseño orientado a objetos, en esta sección se da a conocer el conjunto de clases necesarias para especificar el aspecto temporal en la estructura navegacional en la etapa de diseño. Para cada una de las clases que se presentan en las siguientes subsecciones serán especificadas su composición y su responsabilidad dentro de la arquitectura. Al finalizar la sección se presenta la arquitectura resultante.

Links temporizados

A partir de los ejemplos de aplicaciones web de la sección anterior se pueden identificar dos formas en que el aspecto temporal está presente. Es evidente que en el primer ejemplo el comportamiento que da origen a la modificación de la estructura navegacional sucede solamente una vez a lo largo de la vida útil de la aplicación mientras que en el segundo la misma modificación sobre tal estructura puede repetirse varias veces. En consecuencia, para especificar ambos comportamientos en la etapa de diseño proponemos dar origen a una jerarquía de links temporizados. Esta jerarquía se muestra en la figura 8.3 y estará formada por una superclase denominada `TemporalLink` con subclases `FixedTemporalLink` e `IterativeTemporalLink`.

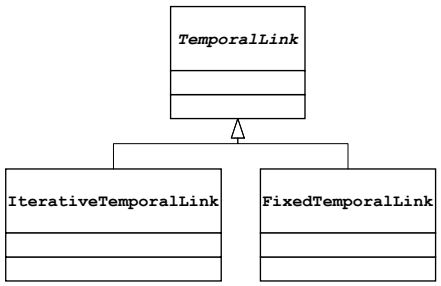

Figura 8.3: *Jerarquía de links temporizados.*

De esta manera, la responsabilidad de la subclase `FixedTemporalLink` será especificar el momento en que la estructura navegacional será modificada. Por lo contrario, la responsabilidad de la subclase `IterativeTemporalLink` será especificar la frecuencia y durante cuanto tiempo se repetirá una misma modificación sobre la estructura de navegación. Más adelante se especificarán las demás clases colaboradoras de ésta última.

Acciones

Siguiendo con los ejemplos citados, otro punto a tomar en cuenta es que no solamente se necesita saber el momento en que sucede el deadline sino también cual será la nueva forma que tomará la estructura navegacional de la aplicación. En consecuencia, lo que proponemos es desacoplar este comportamiento temporal en dos componentes principales: la acción o algoritmo que realizará la modificación y la referencia temporal en la cual la acción se llevará a cabo.

Por tanto, para modelar la primer componente planteamos la definición de una clase abstracta

`Action` de manera tal que defina un protocolo a ser reimplementado por el diseñador de la aplicación. La idea principal del diseño de esta jerarquía de clases es utilizar el modelo del patrón de diseño Strategy [GHJV94] para que el diseñador de la aplicación defina un modelo personalizado a cada tipo de modificación. La figura 8.4 muestra la definición de esta jerarquía.

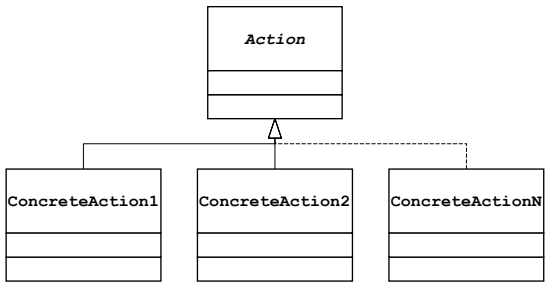

Figura 8.4: *Jerarquía de acciones.*

Con la definición de esta jerarquía de clases, las dos interpretaciones que fueron descritas en la primer sección de este capítulo podrán ser especificadas como subclases de la clase abstracta `Action` con la reimplementación del comportamiento correspondiente.
Para modelar la segunda componente del comportamiento temporal proponemos el empleo de una jerarquía de clases compuesta por una superclase `TimeReference` y las subclases `Point` e `Interval` como se muestra en la figura 8.5.

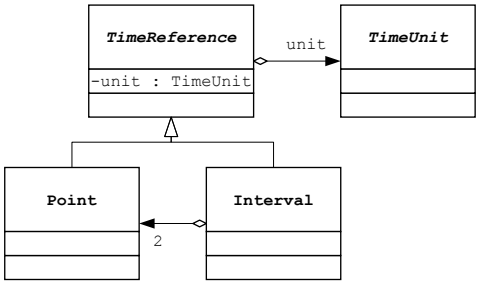

Figura 8.5: *Jerarquía de referencias temporales.*

Con la definición de esta jerarquía de clases, la responsabilidad de la subclase `Point` será modelar un punto específico en el tiempo mientras que la responsabilidad de la subclase `Interval` será modelar un intervalo temporal compuesto por un punto de tiempo inicial y un punto de tiempo final. Así, con la primera subclase se podrán especificar referencias temporales fijas como por ejemplo la fecha que

señala el deadline en el primer ejemplo del capítulo anterior mientras que con la subclase `Interval` se podrán obtener referencias temporales que especifiquen periodos de tiempo como por ejemplo las temporadas de venta a bajo precio del segundo ejemplo del anterior capítulo. Los valores atribuidos a las instancias de ambas subclases serán interpretados de acuerdo a la unidad de tiempo asociada. La figura 8.6 muestra un diagrama de clases donde se componen ambas jerarquías recientemente descriptas.

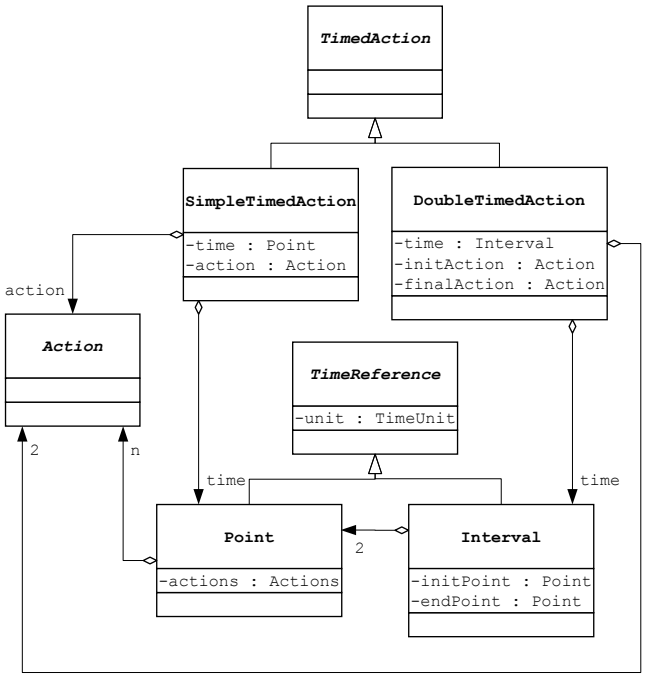

Figura 8.6: *Composición de referencias temporales con acciones.*

En la figura anterior se puede observar que para cada referencia de tiempo se asocia una acción determinada. A partir de este diagrama de clases, en la práctica será fácil obtener composiciones acción-referencia de manera que en un mismo instante de tiempo se realicen simultáneamente diferentes modificaciones sobre la estructura navegacional de la aplicación.

Iteradores

Al presentar la jerarquía de links temporizados, se mostró que la subclase `IterativeTemporalLink` especifica tanto la frecuencia como la cantidad de veces en que se modifica la estructura navegacional. En algunas aplicaciones, la modificación de una estructura en el tiempo puede formar parte de la

aplicación en sí; es decir que no sucede una cantidad limitada de veces. Por este motivo proponemos el empleo de una jerarquía de iteradores como la que se muestra la figura 8.7.

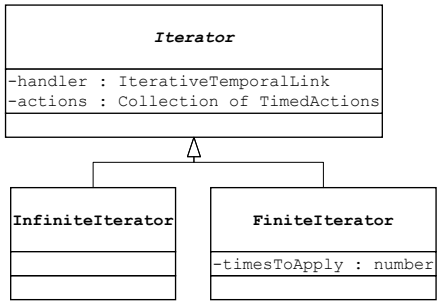

Figura 8.7: *Jerarquía de iteradores.*

La responsabilidad que poseen en común ambos iteradores es realizar la ejecución de la acción que modifica la estructura navegacional y luego asignar a una instancia de la clase `TemporalLink` con la siguiente acción. En particular, la subclase `InfiniteIterator` especifica que este comportamiento se realice indefinidamente durante toda la vida útil de la aplicación. En cambio, la subclase `FiniteIterator`, solamente especifica tal comportamiento un número o cantidad fija de veces mediante su atributo `timesToApply`. A partir de todas las clases presentadas en las subsecciones anteriores se puede formar una arquitectura orientada a objetos útil para especificar aspectos temporales en la estructura navegacional de una aplicación hypermedia. La arquitectura final se puede observar en la figura 8.8.

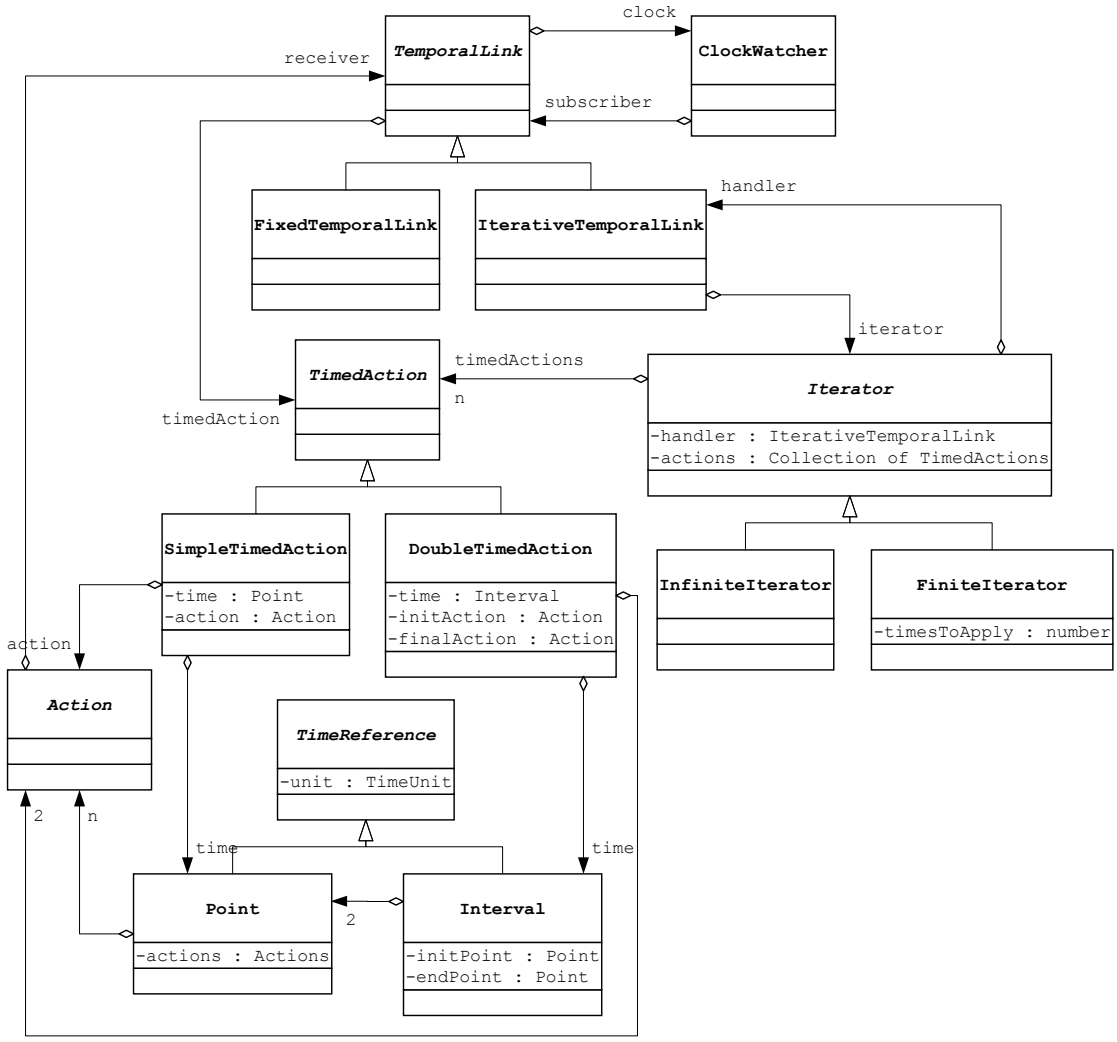

Figura 8.8: *Arquitectura para la especificación del aspecto temporal en la estructura navegacional.*

En la figura de la arquitectura se observa la aparición de una clase `ClockWatcher`. La referencia a ésta es solamente a manera de especificar la implementación de un objeto que permita hacer scheduling de tiempo sobre las acciones especificadas. En la etapa de implementación se debe tomar en cuenta que la tecnología debe soportar los elementos necesarios para definir estos objetos.

Capítulo 9

Ejemplos de Aplicación

Este capítulo presenta ejemplos de uso de las arquitecturas desarrolladas en los capítulos 6 y 8. El primero de ellos se basa en el desarrollo de una aplicación web que muestra la colección de obras producidas por Cândido Portinari [LMP+93] mientras que el segundo ejemplo se basa en el desarrollo de una aplicación web perteneciente al área del e-commerce. Dado que el objetivo de este trabajo no es mostrar como se debe utilizar OOHDM, se supondrá que tanto el esquema conceptual como el esquema navegacional fueron desarrollados con anterioridad. Además, también se supondrá que en tiempo de ejecución no existe ningún error de transmisión de datos y que todos los elementos multimedia están a disposición en el momento necesario. A continuación se muestra un ejemplo de especificación de aspectos temporales entre elementos multimedia de la interfaz del usuario y luego un ejemplo de especificación de aspectos temporales en la estructura navegacional de la aplicación.

9.1 Especificación en la Interfaz del Usuario

Esta sección presenta tres ejemplos de especificación de aspectos temporales en la interfaz del usuario. Todos toman como domino de aplicación la especificación de la interfaz de un nodo que contiene tres entidades multimedia a presentar: un video que muestra como llegar a una determinada sala de un museo, una secuencia de imágenes que muestra todas las pinturas exhibidas en esa sala, un texto que describe las características comunes de cada una de ellas, y un botón para iniciar la reproducción del video.

En primer lugar se supondrá un escenario en el cual la prestación del texto y del botón es permanente, y que la presentación de la secuencia de pinturas se realiza inmediatamente a continuación de la presentación del video. La figura 9.1 muestra la composición de ADVs de la interfaz del nodo y la secuencia de ejecuciones.

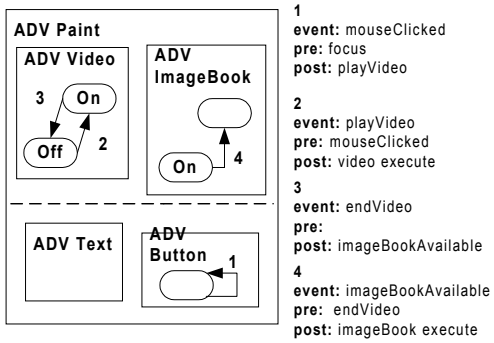

Figura 9.1: *Especificación de la interfaz de un nodo usando ADVs.*

La composición de ADVs es correcta, pero en caso de que se precise especificar de que manera se realiza la sincronización que poseen los elementos de la secuencia de pinturas es necesario emplear un mecanismo adicional. Por tanto, se emplearan instancias de la arquitectura *Intra Frame Synchronization* de modo que utilicen las synchronization constrains adecuadas para especificar la sincronización requerida. De esta manera, si las pinturas deben presentarse una a continuación de otra y sin tiempo de demora entre ellas se puede realizar la especificación que muestra la figura 9.2.

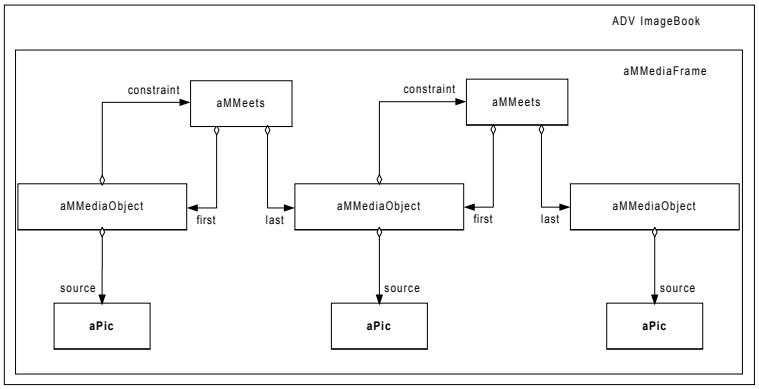

Figura 9.2: *Especificando la presentación de la secuencia de pinturas.*

De otra forma, si las pinturas presentarse una a continuación de otra con un tiempo de demora entre ellas, simplemente se reemplazan todas o algunas de las instancias de la clase `MMeets` por instancias de la clase `MAfter`.

Ahora supongamos un nuevo escenario en el que la presentación de la secuencia de pinturas debe empezar unos instantes más tarde de que termine la presentación del video. Es evidente que la especificación de ADVs de la figura 9.1 tampoco es de gran utilidad. Si bien ésta expresa que el fin de la presentación del video da origen a la presentación de las imágenes, no dice nada con respecto al tiempo de demora entre presentaciones. Por lo tanto, para poder especificar este nuevo escenario realizaremos una instanciación de la arquitectura *Inter Frame Synchronization* como lo muestra la figura 9.3.

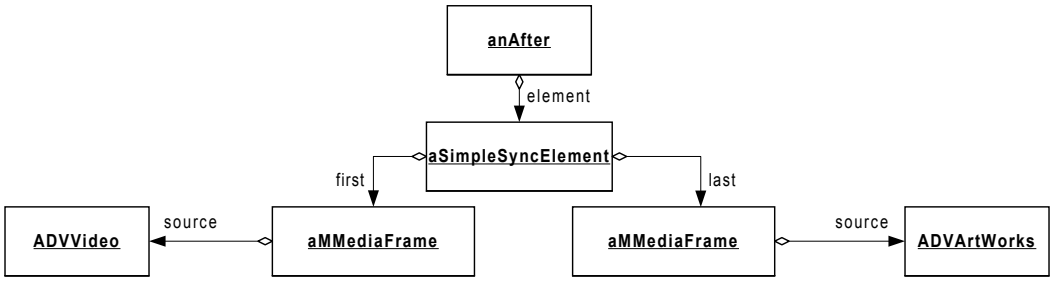

Figura 9.3: *Especificando la sincronización entre la presentación del video y secuencia de pinturas.*

En la figura se puede observar que ADVVideo y ADVArtwork se componen mediante una instancia de la clase `After`. Este objeto señala la existencia de una sincronización de tipo secuencial y el valor de su variable de instancia `delayFor` marca el tiempo de demora entre ambos ADVs. Si la presentación de la secuencia de pinturas debe realizarse inmediatamente a continuación de la presentación del video, entonces solamente es necesario cambiar la instancia de la clase `After` por una instancia de la clase `Meets`.

Ahora supongamos otro escenario un poco más complejo en el que la presentación de la secuencia de pinturas debe realizarse a continuación de la presentación del video y la presentación del texto durante la presentación de la secuencia de pinturas. La especificación de este escenario lo muestra la figura 9.4.

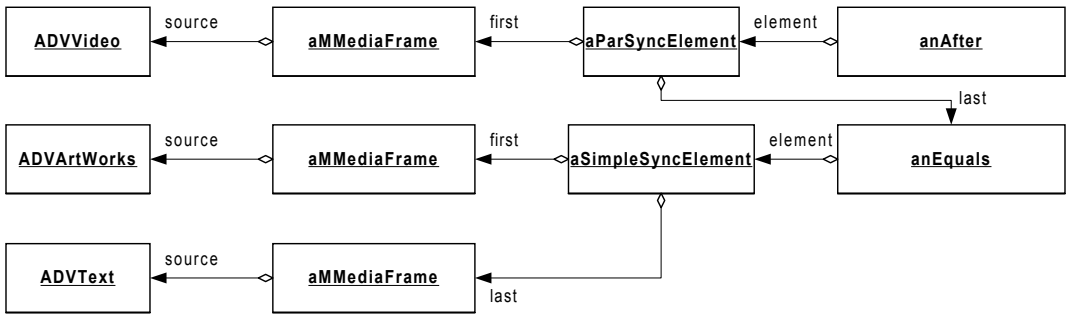

Figura 9.4: *Especificando la sincronización entre la presentación del video, la secuencia de pinturas y el texto.*

En este caso se puede observar el empleo de una instancia de la clase `After`. Este objeto especifica sincronización entre la presentación de ADVVideo y un grupo de entidades sincronizadas formado por ADVArworks y ADVText. El tipo de sincronización que existe entre las presentaciones de estos dos últimos elementos se establece por una instancia de la clase `Equals`.

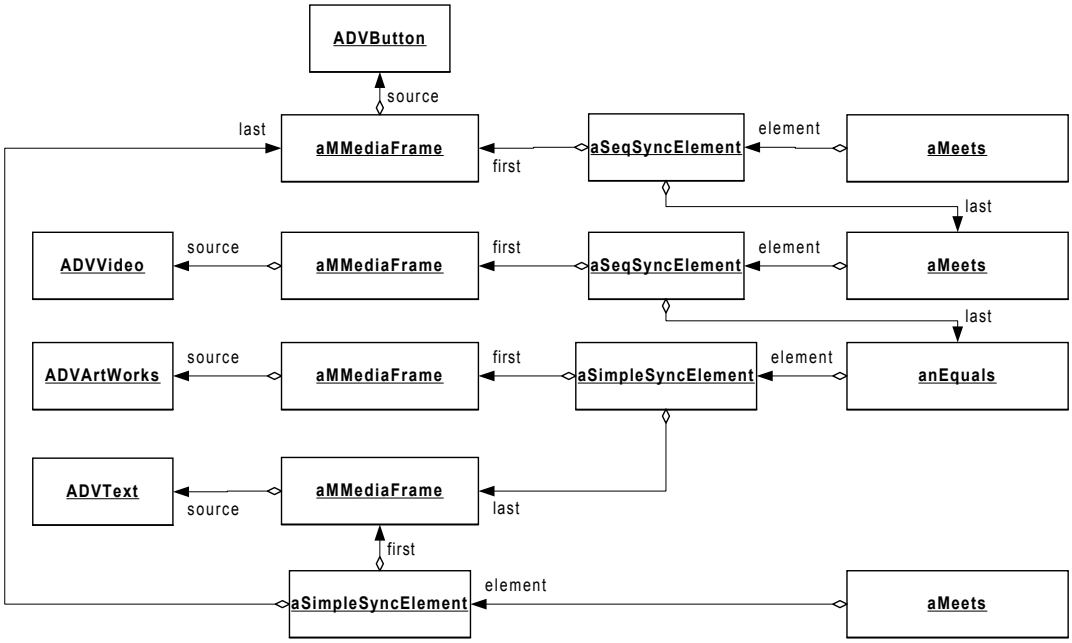

Figura 9.5: *Especificando la sincronización entre la presentación del video, la secuencia de pinturas, el botón y el texto.*

Como último ejemplo, se supongamos un escenario en el cual el botón que da inicio a la presentación del video debe presentarse sólo cuando el video y las pinturas no se están presentando. En otras palabras, el botón solamente debe estar visible cuando no se presenta ADVVideo o ADVArtworks. Una manera de especificar este escenario sería sincronizar la presentación del botón con respecto a la presentación del video y también con respecto a la presentación del texto o de la secuencia de figuras. La especificación completa lo muestra la figura 9.5.

De este modo se puede observar que ADVButton interactúa con dos objetos de sincronización. El primero es un objeto de la clase `Meets` que especifica la sincronización entre las presentaciones del ADVButton y el ADVVideo. Esta instancia define que primero se realizara la presentación del botón e inmediatamente a continuación la presentación del video. Se puede suponer que el fin de la presentación del botón está marcado cuando el usuario ejecuta una acción que dispara un evento asociado y se desactiva alguna propiedad como el atributo visible que proveen los objetos de HTML. El segundo objeto con quien interactúa ADVButton es una instancia de la clase `After`. Este objeto especifica que primero se realiza la presentación de ADVText y luego de un instante la presentación del ADVButton. El valor de ese instante o demora entre el fin de la primer presentación y el comienzo de la segunda queda definido con el valor de la variable de instancia `delayFor`.

9.2 Especificación en la Estructura Navegacional

Esta sección presenta dos ejemplos de especificación de aspectos temporales en la estructura de navegación de la aplicación. Ambos ejemplos toman como dominio de aplicación el sector de ventas de un sitio web que se dedica al comercio de libros. Dado que el objetivo de este trabajo no es realizar una aplicación de OOHDM, en delante se supondrá que tanto el esquema conceptual como el esquema navegacional fueron previamente definidos. La figura figura 9.6 muestra la sección del esquema navegacional sobre el cual se desarrollan los ejemplos.

Como primer ejemplo se supongamos que el comercio esta habilitado para vender productos `DayBook` y `Book` de los cuales el primero solamente se vende en una determinada fecha y el segundo en cualquier momento. Para especificar este escenario realizamos una instancia de la arquitectura propuesta en el capítulo 8 y se la integramos al esquema navegacional de la manera en que lo muestra la figura 9.7. Dado que la venta de `DayBook` se realiza durante un periodo, la instancia de esta arquitectura emplea un `FixedTemporalLink` con una acción asociada subclase de `Action` denominada `DailySales`. Esta

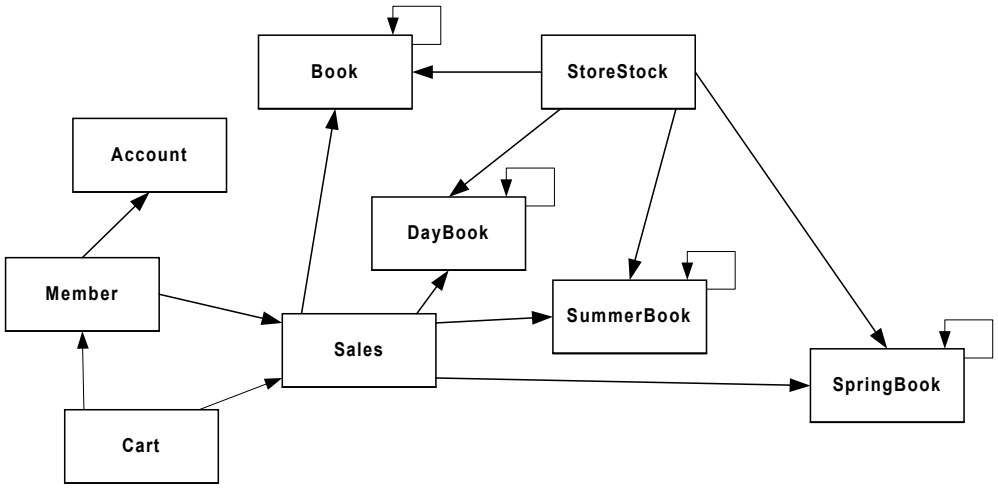

Figura 9.6: *Parte del esquema navegacional de la aplicación de comercio electrónico.*

subclase especifica el momento durante el cual está vigente la venta del producto `DayBook`. Además, también permite modificar la estructura navegacional cuando sea necesario de manera que el usuario tenga habilitado el enlace para realizar compras o browsing sobre los productos mencionados.

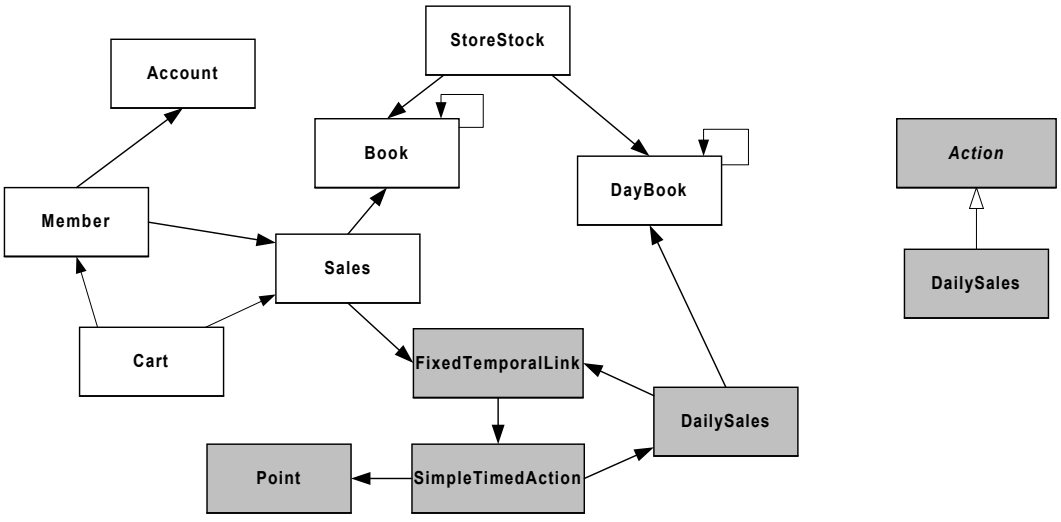

Figura 9.7: *Especificando la venta de* `DayBook`.

En el ejemplo se puede observar que el resultado de ésta especificación permite que la arquitectura se modifique en dos puntos del tiempo. La primer modificación se realiza cuando sucede la fecha

especificada en que se pueden comprar artículos de clase `DayBooks`. Hasta ese instante el usuario sólo tiene permitida la navegación sobre `Book` y luego recién se le permite el acceso hacia otra colección de libros. La segunda modificación surge cuando la fecha de venta de los productos `DayBook` expira. A partir de ese momento el usuario ya no tiene acceso a los artículos `Daybook` y el esquema navegacional de la aplicación vuelve a tener la misma estructura que antes.

Ahora se supongamos un nuevo escenario donde se debe especificar la venta de tres clases de libros durante la duración de tres intervalos de tiempo distintos. En éste caso, los productos de la clase `Book` sólo se venden en otoño e invierno, los productos de `SpringBook` en primavera y los `SummerBook` en verano. Además, se supondrá que esta configuración de ventas se repite anualmente durante toda la vida útil de la aplicación. La figura 9.8 muestra la especificación del esquema navegacional de acuerdo al escenario establecido empleando la arquitectura propuesta.

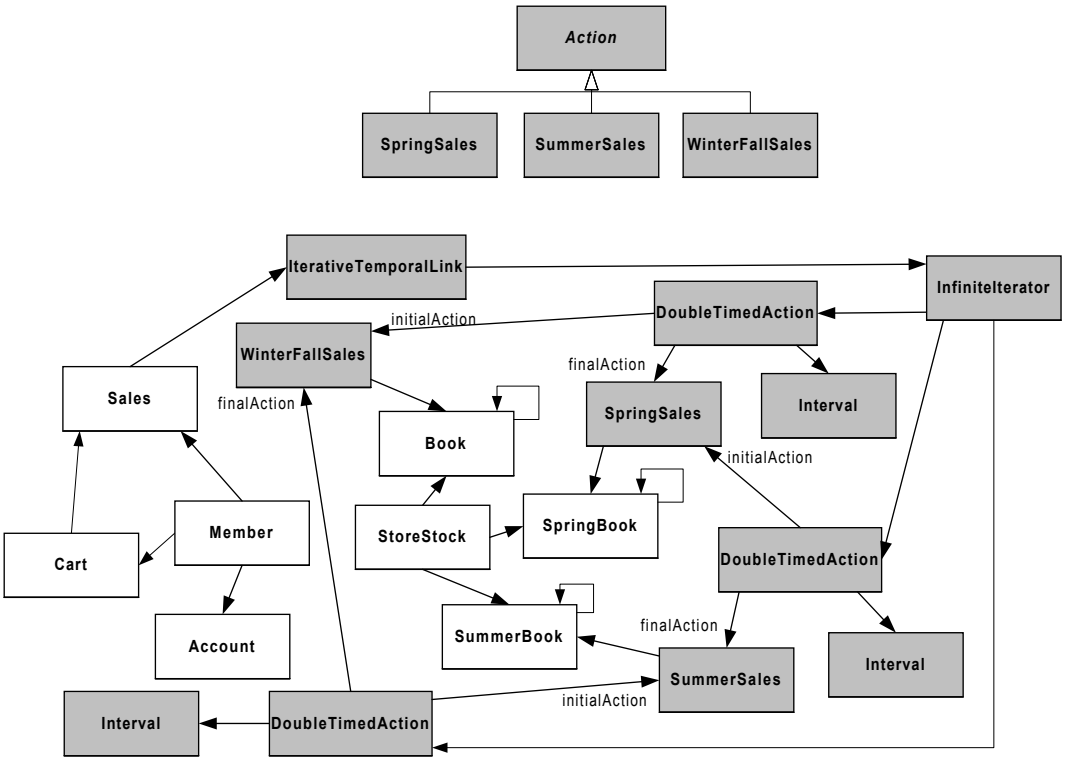

Figura 9.8: *Especificando la venta de `Book`, `SpringBook` y `SummerBook`.*

En este caso, la instanciación de la arquitectura define los tres intervalos de tiempo mediante las subclases de `Action` denominadas `WinterFallSales`, `SpringSales` y `SummerSales`. Estas colaboran con su correspondiente `DoubleTimedAction` e implementan las acciones a llevar a cabo durante cada intervalo. Se puede ver que `WinterFallSales` es la responsable de realizar las acciones hasta que suceda `SpringSales`, luego ésta es la responsable hasta que suceda `SummerSales` y, finalmente, ésta responderá por las acciones a llevar a cabo hasta que el fin de su responsabilidad lo marque `WinterFallSales`. La especificación emplea un `InfiniteIterator` lo que significa que la secuencia de responsabilidades recientemente descrita se repetirá infinitamente. Si por el contrario, la secuencia debe realizarse una cantidad finita de veces, entonces se reemplaza `InfiniteIterator` por `FiniteIterator` y se valoriza su variable de instancia `timesToApply`.

En este ejemplo se modifica tres veces el esquema navegacional en tres puntos del tiempo. La primer modificación se realiza cuando sucede el inicio de la temporada de `SpringBooks`; hasta ese instante el usuario solo tiene permitida la navegación hacia `Books`. La segunda modificación surge cuando se inicia la temporada de `SummerBooks`; hasta ese momento el usuario solo tiene permitido navegar sobre `SpringBooks`. Por último, la tercer modificación veda al usuario los recorridos sobre `SummerBooks` y rehabilita el acceso a `Books`.

9.3 Ejemplo Adicional

Esta sección presenta un nuevo ejemplo de aplicación. En esta ocasión, el objetivo será realizar la especificación de una aplicación web que modele el funcionamiento de un cine. Para enriquecer el dominio de la aplicación, consideraremos que el cine consta de tres salas en las cuales se proyectan dos películas por semana de acuerdo a un cronograma de funciones preestablecido. La figura 9.9 muestra una posible instanciación del cronograma de películas que se proyectan en cada sala.

Sala 1	**Sala 2**	**Sala 3**
Analízate Domingo a Martes 15.05; 17.50; 20.35 **Simone** Miércoles a Sábado 15.05; 17.50; 20.35	**El libro de la Selva 2** Domingo a Miércoles 16.20; 19.40; 23.00 **Piso Compartido** Jueves a Sábado 16.20; 19.40, 23:15	**Mini Espías** Domingo a Martes 16.20; 19.40; 23.00 **8 Mile** Miércoles a Sábado 16.20; 19.40; 23:15

Figura 9.9: *Cronograma de funciones del cine.*

Nuevamente, para desarrollar el diseño de la aplicación emplearemos la metodología OOHDM y, de la misma forma que en el ejemplo de la sección anterior, supondremos que tanto el esquema conceptual como el esquema navegacional ya fueron realizados. La figura 9.10 muestra un posible diseño de este último.

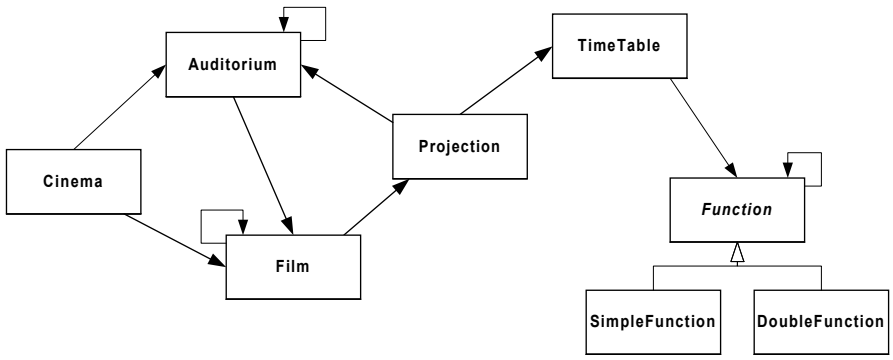

Figura 9.10: *Esquema navegacional del cine.*

De acuerdo al esquema navegacional, se puede observar que el usuario de la aplicación puede visitar cada una de las salas del cine y desde ésta conocer la película que proyecta. Tomando en cuenta un escenario un poco más realista y adaptado a los requerimientos que plantea el cronograma de funciones, cada vez que el usuario visita una sala debería poder ver información de la película asociada a la función estipulada para ese momento. Esto es, si el lunes visita la Sala 1 entonces la aplicación debería mostrar información relacionada a la película Analízate. Mientras que si visita la misma sala el día jueves, entonces la aplicación debería mostrar información relacionada a Simone.

Este análisis nos revela la existencia de un aspecto de características netamente temporales en el dominio de la aplicación. Dicho aspecto señala que, durante la semana, la aplicación debe cambiar la información de cada sala que brinda a los usuarios de la misma forma y en el mismo momento en que cambian las películas que éstas proyectan. Desde el punto de vista de diseño significa que el tramo del esquema navegacional que permite navegar hasta la película que se proyecta en una sala debe mudar tantas veces y en el mismo momento en que se realizan los cambios de las películas. Siguiendo con el ejemplo de la Sala 1, el primer cambio se debe realizar cuando se permuta la película que se proyecta de domingo a martes por la película que se proyecta de miércoles a viernes. Mientras que el segundo cambio debe suceder cuando se permuta esta última por la que se proyecta de domingo a martes.

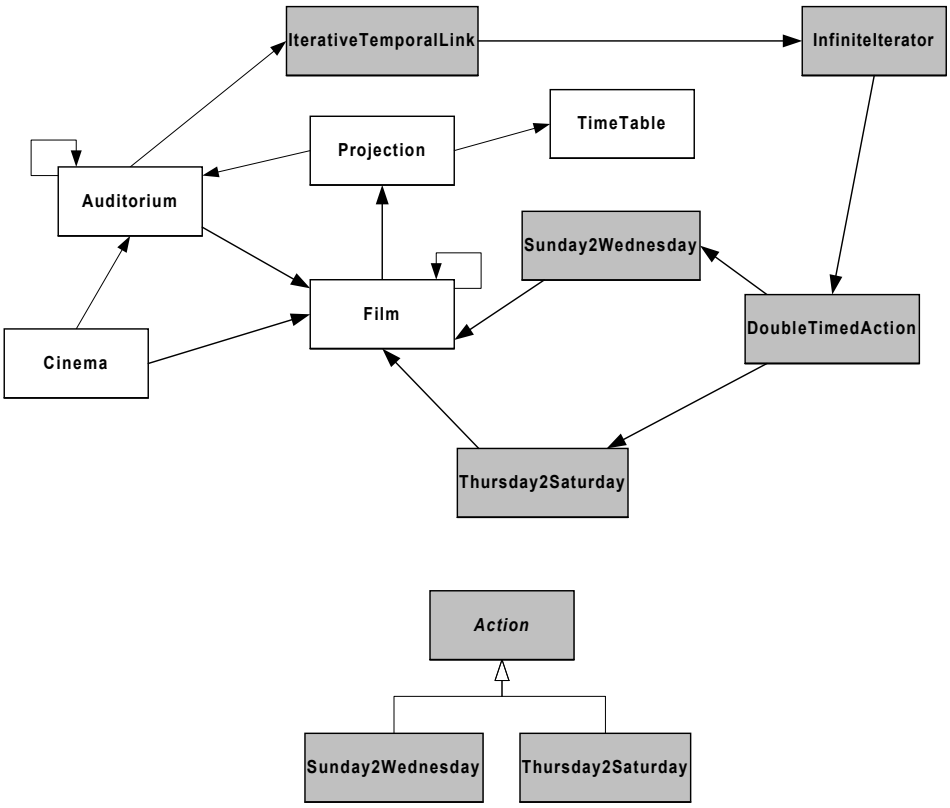

Figura 9.11: *Esquema navegacional del cine con especificación de tiempo.*

Una vez más, para poder especificar en la etapa de diseño el aspecto temporal que aparece en la estructura navegacional de la aplicación necesitamos emplear algún mecanismo de especificación de tiempo. Por este motivo, pondremos en práctica las arquitecturas de especificación de tiempo propuestas en el capítulo 8. Así, se obtiene un nuevo esquema navegacional como lo muestra la figura 9.11. En el esquema navegacional modificado se puede observar que la instanciación de la arquitectura modela para cada sala dos intervalos de tiempo mediante las subclases de `Action` denominadas `Sunday2Wednesday` y `Thursday2Saturday`. Estas colaboran con su correspondiente `DoubleTimedAction` e implementan las acciones a llevar a cabo durante cada intervalo semanal. Se puede ver que `Sunday2Wednesday` es la responsable de realizar las acciones hasta que suceda `Thursday2Saturday`. La especificación emplea un `InfiniteIterator` lo que significa que el esquema de cambios de acciones y la secuencia de responsabilidades recientemente descrita se repetirá mientras perdure la aplicación. Si por el contrario, la secuencia debe realizarse una cantidad finita

de veces, entonces se reemplaza `InfiniteIterator` por `FiniteIterator` y se valoriza su variable de instancia `timesToApply`.

Ahora, supongamos un nuevo escenario en el cual el cronograma de funciones es un poco más complejo. Consideremos que, además de que cada sala tiene dos películas que se proyectan de manera alternada durante la semana, también se proyectan en un mismo día en diferentes horarios. En la figura 9.12 se muestra un ejemplo del cronograma de funciones modificado con este nuevo requerimiento.

Sala 1	**Sala 2**	**Sala 3**
Analízate Domingo a Martes 15.05; 17.50; 20.35 Sábado 17.50; 20.35 **Simone** Miércoles a Viernes 15.05; 17.50; 20.35 Sábado 23.20	**El libro de la Selva 2** Domingo a Miércoles 16.20; 19.40; 23.15 Sábado 16.20; 19.40 **Piso Compartido** Jueves y Viernes 16.20; 19.40, 23:15	**Mini Espías** Sábado a Martes 16.20; 19.40; 23.00 **8 Mile** Miércoles a Viernes 16.20; 19.40; 23:15 Sábado 16.20; 19.40

Figura 9.12: *Nuevo cronograma de funciones del cine.*

De acuerdo a la nueva cartelera, se puede observar que el dominio de la aplicación contiene un nuevo aspecto con características temporales. Este aspecto señala que, durante el mismo día, la aplicación debe cambiar la información de cada sala que brinda a los usuarios de la misma forma y en el mismo momento en que cambian las películas que éstas proyectan. Desde el punto de vista de diseño de la aplicación significa que el tramo del esquema navegacional que permite al usuario visitar la película que se proyecta en una sala debe mudar de forma tantas veces y en el mismo momento en que se realizan los cambios de las películas. Siguiendo con el ejemplo de la Sala 1, el primer cambio se debe realizar de martes a miércoles cuando se permuta la película Analízate por Simone. El segundo cambio cuando reaparece Analízate durante en las primeras funciones del sábado. Luego, el tercer cambio sucede cuando Simone reemplaza en la trasnoche a Analízate y, finalmente, el último cambio cuando Analízate nuevamente vuelve a ser la función actual de la sala a partir del domingo.

Por tanto, empleando las arquitectura de especificación de tiempo propuestas en el capítulo 8, se obtiene un nuevo esquema navegacional como lo muestra la figura 9.13. En este nuevo esquema navegacional se puede observar que la instanciación de la arquitectura define para cada sala tres intervalos de tiempo mediante las subclases de Action denominadas `Sunday2Wednesday`,

`Thursday2Friday` y `Saturday` implementando así las acciones a llevar a cabo durante cada intervalo semanal. Se puede ver que `Sunday2Wednesday` es la responsable de realizar las acciones hasta que suceda `Thursday2Friday` y esta misma se encarga de redireccionar las acciones de navegación del usuario hasta que suceda `Saturday`. Por otro lado, las funciones que forman el cronograma se subclasifican en dos tipos de funciones `DoubleFunction` y `SimpleFunction`.

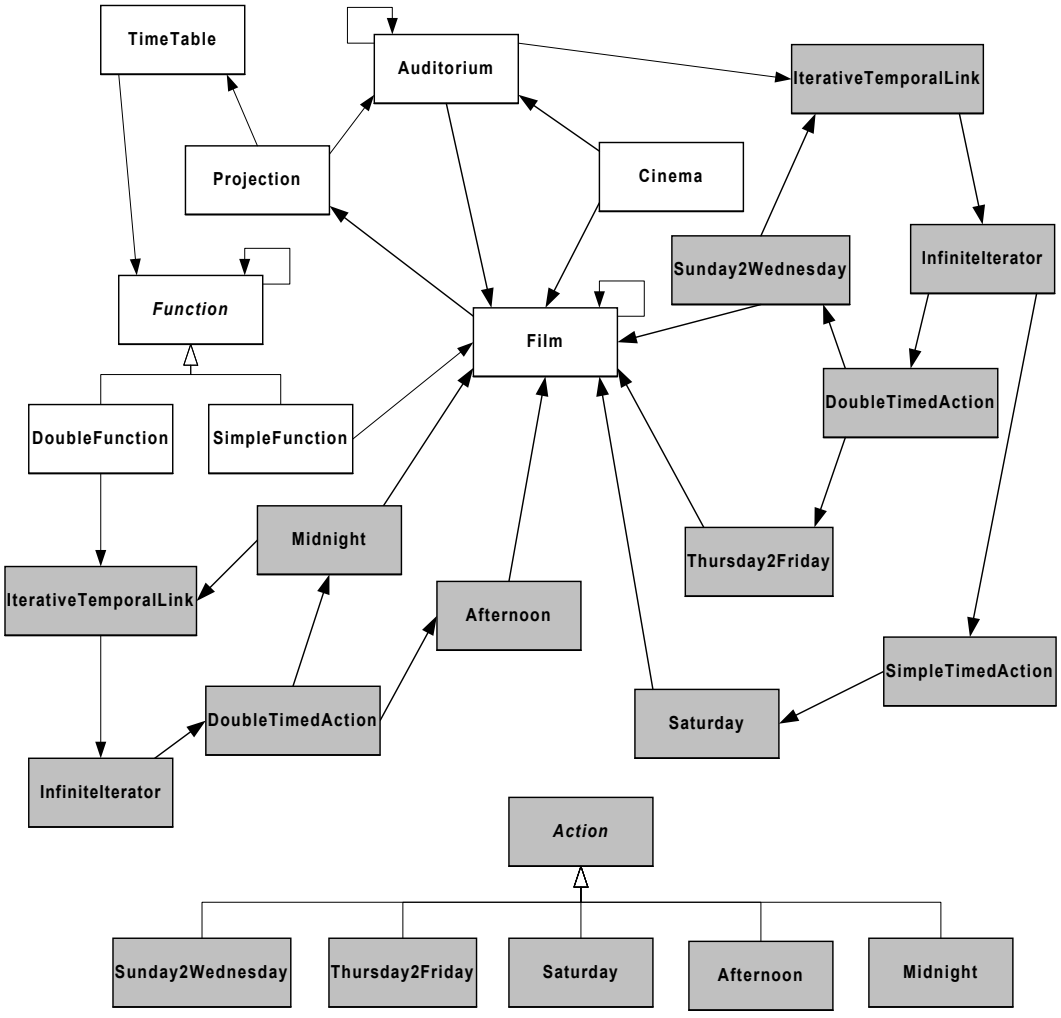

Figura 9.13: *Nuevo esquema navegacional del cine con especificación de tiempo.*

La primer clase de esta jerarquía cuenta con la responsabilidad de interactuar con otra instancia de la arquitectura de especificación de aspectos temporales de manera que todo en conjunto modele el aspecto temporal agregado en este nuevo escenario.

Hemos visto que así como existen aspectos temporales en la estructura navegacional de la aplicación, éstos también están presentes en la interfaz del usuario. Por ejemplo, supongamos que la interfaz del nodo sala del esquema navegacional se compone de un video que muestra los avances de las películas que la sala proyecta, un texto que muestra las características de esa película como por ejemplo la duración, el director, los actores y la procedencia, y un listado con los horarios y títulos de las demás películas que se proyectaran en la semana. Para especificar la manera en que éstas tres entidades se componen y comunican, podemos emplear un diagrama de ADVs de la manera en que lo muestra la figura 9.14.

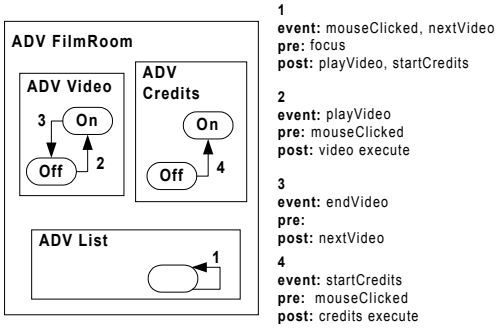

Figura 9.14: *Especificación de la interfaz del usuario de una sala.*

Esta composición de ADVs muestra de manera abstracta la relación existente entre las entidades que forman la interfaz del usuario. Sin embargo, no dice nada sobre una posible coordinación temporal que pueda existir entre ellas. Por ejemplo, si cada vez que se inicia la proyección del avance de una película se debe demorar la presentación de los créditos, entonces la especificación con ADVs no ayuda. Por tanto, sugerimos emplear la arquitectura *Interframe Synchronization* propuesta en el capítulo 6. La figura 9.15 muestra la especificación de este escenario temporal en la interfaz del usuario.

La figura anterior define que la instancia de la clase `Equals` especifica que la presentación de ADVList y ADVVideo siempre sucede en paralelo y que la clase `After` especifica que la prestación de

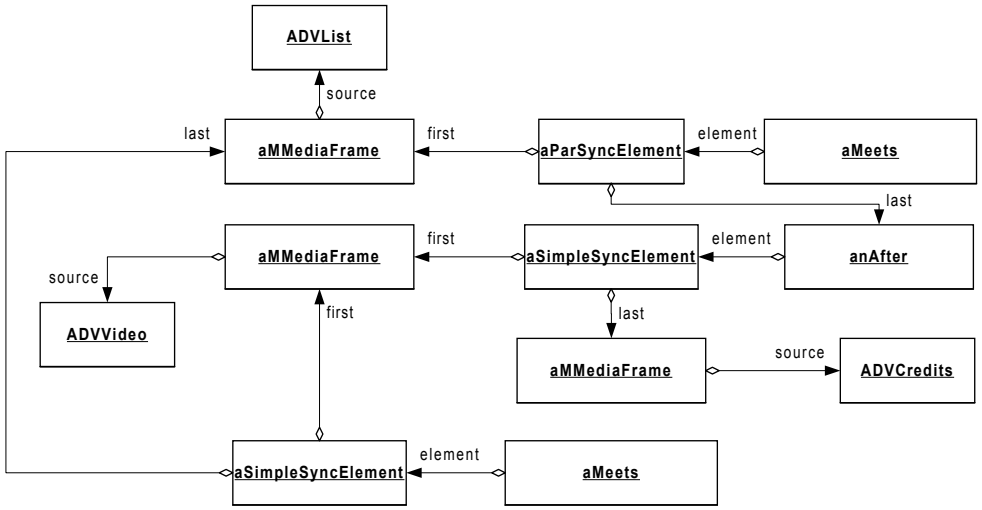

Figura 9.15: *Especificando sincronización entre la presentación del video, los créditos y la lista de películas.*

ADVVideo y ADVCredits se realiza en secuencia con una demora entre ambas presentaciones. El valor de la demora debe estar especificado en la variable de instancia `delayFor` de la instancia de clase `After`.

Para mostrar una nueva aplicación de los mecanismos definidos analizaremos otro escenario distinto. En esta ocasión supongamos que cada vez que el usuario selecciona el nombre de una película de la lista ésta desaparece dando paso a la presentación del avance seleccionado y luego que éste termina la lista vuelve a estar presente en la interfaz del usuario. Esta especificación se muestra en la figura 9.16.

En esta especificación se observa que el empleo de una instancia de clase `Meets` basta para especificar que la presentación de ADVList y ADVVideo sucedan en ese orden de manera consecutiva y sin demoras. También e puede observar que la instancia de clase `After` especifica que las presentaciones de ADVVideo y ADVCredits se realicen en secuencia con una demora entre ambas presentaciones y, finalmente, que la instancia de clase `Meets` especifica que luego de la presentación del ADVVideo suceda la presentación de ADVList. En este último ejemplo se puede notar que si bien la primer instancia de `Meets` especifica lo mismo que señala la secuencia de eventos que especifica la figura 9.14, el empleo de la arquitectura flexibiliza el diseño y además favorece al mantenimiento de la aplicación. Por ejemplo, si se necesita que entre la presentación del ADVList y el ADVVideo exista

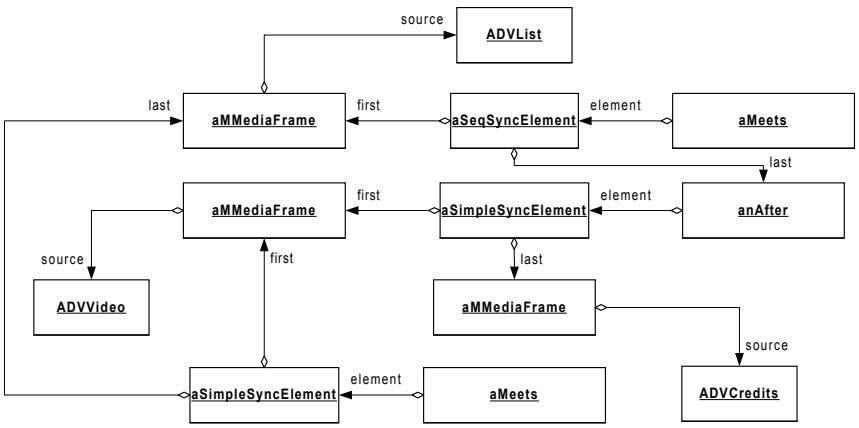

Figura 9.16: *Especificando otra sincronización entre la presentación del video, los créditos y la lista de películas.*

una demora, entonces solamente será necesario cambiar la instancia de `Meets` por una instancia de clase `After` y valorizar su variable de instancia `delayFor`.

En este capítulo se mostraron varios ejemplos de aplicación de las arquitecturas propuestas en los capítulos anteriores. Con cada ejemplo se mostró que el empleo de las arquitecturas propuestas ayudan a modelar los aspectos temporales de la estructura navegacional y de la interfaz del usuario de manera simple, proveen un modelo fácil de entender, agregan semántica a la especificación y, por ser modelos orientados a objetos, el diseño resultante es simple de extender y de mantener.

Capítulo 10

Conclusiones y Trabajo Futuro

En este trabajo se propusieron dos arquitecturas para especificar los aspectos temporales de una aplicación hypermedia en la etapa de diseño. El desarrollo de ambos modelos abarcó dos etapas: especificación del tiempo en la interfaz del usuario y especificación del tiempo en la estructura navegacional de la aplicación.
La primer etapa tuvo como fundamentos el análisis de los diferentes modelos teóricos que existentes para representar el tiempo y las características de los escenarios temporales que ocurren durante la presentación de los elementos multimedia que componen la interfaz del usuario de las aplicaciones hypermedia. A partir de ambos se definió una arquitectura que, mediante composición y herencia de clases, permite especificar una variedad de modelos de tiempo en la etapa de diseño de la interfaz del usuario de una aplicación hypermedia. Por otro lado, la segunda etapa tuvo como fundamentos la detección y el análisis de escenarios temporales que ocurren en la estructura navegacional de las aplicaciones hypermedia. A partir de éstos se propusieron diferentes enfoques tentativos capaces de interpretar el comportamiento y, sobre la elección de uno de ellos, se definió una arquitectura compuesta por jerarquías de clase y relaciones entre objetos capaces de definir diferentes modelos de tiempo en la etapa de diseño de la estructura navegacional de una aplicación hypermedia.

Dado que las arquitecturas propuestas se basan en un modelo orientado a objetos, éstas heredan todos los aspectos y las propiedades de un diseño de esas características. Por tal motivo, permiten realizar modificaciones, ampliaciones y mantenimiento de una manera práctica y sencilla. Además, la herencia y el encapsulamiento le permiten al diseñador abstraerse de la implementación de sus componentes en el momento de la especificación de escenarios con aspectos temporales.

Un aspecto importante a determinar en el futuro es establecer la manera y el momento más apropiado para incorporar las arquitecturas propuestas a OOHDM o cualquier metodología de diseño de aplicaciones hypermedia orientada a objetos. En particular, el empleo de ADVs y state charts en OOHDM permiten especificar de manera abstracta el orden en el cual se ejecutan los elementos que conforman la interfaz del usuario. Dado que orden de ejecución no necesariamente incluye sincronización, una alternativa válida sería emplear la arquitectura de especificación de aspectos temporales de la interfaz del usuario en un paso posterior a la etapa de especificación de la interfaz abstracta. De este modo, la definición del orden en que sucede la ejecución de los elementos que forman la interfaz del usuario queda desacoplada respecto de la manera en que estos se sincronizan. Esto tiene la ventaja de que sobre la base de un orden de ejecución es posible establecer cuantas formas de sincronización sean convenientes. Del mismo modo, dado que una vez que se realiza el esquema navegacional de la aplicación recién son conocidos los mapas navegacionales, proponemos emplear la arquitectura de especificación de aspectos temporales de la estructura navegacional en un paso siguiente a la etapa de diseño del esquema navegacional. De esta manera el diseñador de la aplicación tiene un dominio de especificación más restringido que el esquema conceptual. La ventaja que esto provee es que diferentes especificaciones pueden comprender objetos comunes a contextos navegacionales disjuntos quedando la especificación del aspecto temporal ligada a un contexto y no al modelo conceptual de la aplicación.

Cuando una aplicación está en la etapa de producción, es posible que se manifiesten algunos problemas referentes a la plataforma de ejecución. Un ejemplo muy habitual es lo que sucede en las aplicaciones web cuando se transmiten streams de datos desde el servidor hacia los clientes. Dado que este tipo de transmisión se realiza mediante Internet, los errores, demoras innecesarias, o saturación del medio causan efectos negativos sobre la sincronización de las entidades que conforman la interfaz del usuario. Una solución para sortear ese tipo de problemas es emplear un lenguaje de implementación que soporte mecanismos de sincronización asincrónicos.
Las arquitecturas aquí propuestas, en especial la utilizada para la especificación de aspectos temporales en la interfaz del usuario, solamente contemplan la especificación de modelos de sincronización sincrónica. Por tanto, un objetivo importante a alcanzar en el futuro es complementar este trabajo con mecanismos encargados de especificar modelos de sincronización sincrónicos y asincrónicos. Tal vez este objetivo se puede alcanzar extendiendo de las jerarquías de clase, agregando relaciones de conocimiento entre las clases de las arquitecturas o enriqueciendo las mismas.

El desarrollo de software a gran escala exige el empleo de herramientas que realicen la generación automática de código a partir de la etapa de especificación. Esta demanda nos motiva a proponer, como una tarea complementaria, el desarrollo de una herramienta CASE que, a partir de la especificación de las arquitecturas aquí propuestas, genere una implementación escrita en un lenguaje de programación orientado a objetos o en algún lenguaje de descripción de datos. Asimismo, otra tarea interesante es analizar cuan factible sería incorporar éstas arquitecturas en algún framework de especificación de aplicaciones hypermedia. Este enfoque le permitiría al diseñador de aplicaciones abstraerse de las jerarquías de clase y formar escenarios de tiempo reutilizables en distintas aplicaciones.

Bibliografía

[All83] J. F. Allen. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[BHM92] G. Blakowski, J. Hübel, and Mühlhaüser. Tool Support for the Synchronization and Presentation of Distributed Multimedia. *Computer Communication*, 15:611–618, December 1992.

[Bla93] G. Blakowski. Development and Runtime Support for Distributed Multimedia Applications, 1993.

[BS96] G. Blakowski and R. Steinmetz. A Media Synchronization Survey: Reference Model, Specification, and Case Studies. *IEEE Journal on Selected Areas in Communications*, 14(1):5–35, January 1996.

[BZ93] M. C. Buchanan and P. T. Zellweger. Automatic Temporal Layout Mechanisms. *1st ACM International Conference on Multimedia*, pages 341–350, August 1993.

[CILS93] D. D. Cowan, R. Ierusalimschy, C. J. P. Lucena, and T. M. Stepien. Abstract Data Views. *Structured Programming*, 14(1):1–13, January 1993.

[CL95] D. D. Cowan and C. J. P. Lucena. Abstract Data Views: an Interface Specification Concept to Enhance Design for Reuse. *IEEE Transactions on Software Engineering*, 21(3), March 1995.

[CMHCL94] L. M. F. Carneiro, D. D. Cowan M. H. Coffin, and C. J. P. Lucena. *ADVCharts a Visual Formalism for Highly Interactive Systems.* Software Engineering in Human-Computer Interaction. Cambridge University Press, 1994.

[Com89] Apple Computer. *HyperCard stack design guidelines.* Addison-Wesley, 1989.

[Cor90] IBM Corporation. Audio, Visual Connection User's Guide and Authoring Language Reference, Version 1.05. *IBM Form SI5F-7134-02*, August 1990.

[DIMG95] A. Diaz, T. Isakowitz, V. Maiorana, and G. Gilabert. RMCase: A tool to design WWW applications. *World Wide Web Journal*, 1:559–566, January 1995.

[DK96] A. Duda and C. Keramane. Interval expressions - a functional model for interactive dynamic multimedia presentations. *IEEE International Conference on Multimedia Computing and Systems*, 1996.

[DM87] T. Dean and D. McDermott. Temporal Data Base Management. *Artificial Intelligence*, 32:1–55, 1987.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Aritificial Intelligence*, 49:61–95, 1991.

[Ech98] M. Echiffre. Mheg-5 - Aims, Concepts, and Implementations Issues. *IEEE Multimedia*, March 1998.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing. Addison-Wesley Longman Inc., fourth edition, 1994.

[Gib91] S. Gibbs. Composite Multimedia and Active Object. *OOPSLA*, pages 97–112, 1991.

[GPS93] F. Garzotto, P. Paolini, and D. Schwabe. HDM - a model based approach for hypermedia design. *ACM Transactions on Information Systems*, 11:1–26, January 1993.

[Gro89] AFNOR Expert Group. Multimedia Synchronization: Definitions and Model, Input Contribution on Time Variant Aspects and Synchronization in ODA-Extensions. *ISO IEC JTC 1/SC 18/WG3*, February 1989.

[GSG99] M. Gaedke, D. Schempf, and H. Gellersen. WCML: An enabling technology for the reuse in object-oriented Web Engineering. *Poster-Proceedings of the 8th International World Wide Web Conference (WWW8), Toronto, Ontario, Canada*, 1999.

[Ham72] C. L. Hamblin. Instants and Intervals. *Proc. of the 1st. Conference of the Intl. Society for the Study of Time*, pages 324–331, 1972.

[HDH96] W. Hall, H. Davis, and G. Hutchings. *Rethinking Hypermedia: The Microcosm Approach*. Kluwer, 1996.

[HPSS87] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman. On the formal semantics of statecharts. *Proceedings of the Second IEEE Symposium on Logic in Computer Science*, June 1987.

[HvOM+99] L. Hardman, J. van Ossenbruggen, K. Sjoerd Mullender, L. Rutlegge, and D. C. A. Bulterman. Do You Have the Time ? Composition and Linking in Time-based Hypermedia. *ACM Hypertext 99*, pages 189–196, 1999.

[ISB91] T. Isakowitz, E. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 38:34–48, August 1991.

[ISO92] ISO. Information Technology Hypermedia/Time-based Structuring Language (hytime). *ISO/IEC IS 10744*, August 1992.

[JL+98] M. Jourdan, N. Layaïda, , C. Roisin, L. Sabry-Ismail, and L. Tardif. MADEUS an Authoring Environment for Interactive Multimedia Documents. *ACM Multimedia'98*, 1998.

[Lan94] D. Lange. Object-oriented design method for hypermedia information system. *Proceedings of the Twenty Seventh Hawaii International Conference on System Sciences*, 1994.

[Lay96] N. Layaïda. Issues in Temporal Representations of Multimedia Documents, 1996.

[LG90] T. D. C. Little and A. Ghafoor. Synchronization and Storage for Multimedia Objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, April 1990.

[LG91] T. D. C. Little and A. Ghafoor. Spatio-temporal Composition of Distributed Multimedia Objects for value added Networks. *IEEE Computer*, 24(10):42–50, October 1991.

[LG92] T. D. C. Little and A. Ghafoor. Scheduling of Bandwith-constrained Multimedia Traffic. *Computer Communications*, 15(6):381–387, July 1992.

[LMP+93] R. S. G. Lanzelotte, M. P. Marques, M. C. G. Penna, J. C. Portinari, I. D. Ruiz, and D. Schwabe. The Portinari Project: Science and Art team up together to help cultural

projects. *2nd International Conference on Hypermedia and Interactivity in Museums (ICHIM'93)*, September 1993.

[LPL95] T. D. C. Little and M. J. Perez-Luque. A Temporal Reference Framework for Multimedia Synchronization. *Proceedings of Sync'95*, May 1995.

[Lue98] M. C. Luesenbrink. The Moment in Hypertext: A Brief Lexicon of Time. *ACM Hypertext 98*, pages 106–112, June 1998.

[MP93] M.C.Buchanan and P.T.Zellweger. Automatically Generating Consistent Schedules for Multimedia Applications. *Multimedia Syst.*, 1(2):55–67, 1993.

[MS90] S. Miller and L. Schubert. Time Revisited. *Computational Intelligence*, 6:108–118, 1990.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object Oriented Modeling and Design*. Prentice Hall Inc., 1991.

[RD97] F. Rousseau and A. Duda. A Synchronized Multimedia for the WWW. *Available WWW: http://delos.imag.fr/sync-mm*, 1997.

[Sǵ95a] P. Sénac. Hierarchical Time Stream Petri Net: a Model for Hypermedia Systems. *16th International Conference on Application and Theory of Petri Nets*, 1995.

[Sǵ95b] P. Sénac. Modeling Logical and Temporal Synchronization in Hypermedia Systems. *IEEE Journal on Selected Areas in Telecomunication*, 14(1), January 1995.

[SR95a] D. Schwabe and G. Rossi. Building Hypermedia Applications as Navigational Views of Information Models. *Proceedings of the Hawaii International Conference on System Sciences*, Janary 1995.

[SR95b] D. Schwabe and G. Rossi. The Object Oriented Hypermedia Design Method. *Communications of the ACM*, 38:45–46, August 1995.

[SS90] D. Shepherd and M. Salmouy. Extending ISO to Support Synchronization Required by Multimedia Applications. *Computer Communication*, 13:399–406, September 1990.

[Ste90] R. Steinmetz. Synchronization Properties in Multimedia Systems. *IEEE Journal on Selected Areas in Communications*, 8(3):401–412, April 1990.

[SWD95] P. Sénac, R. Willrich, and M. Diaz. Hypermedia Synchronization Modeling: A Case Study. *ED-MEDIA Educational Multimedia and Hypermedia*, pages 18–21, 1995.

[TGD91] D. Tsichritzis, S. Gibbs, and L. Dami. Active Media in Object Composition. *U. of Geneva*, pages 115–132, June 1991.

[WDG95] R. Weiss, A. Duda, and D. Gifford. Composition and Serch with a Video Algebra. *IEEE Multimedia*, 2(1):12–25, 1995.

[WR94] T. Wahl and K. Rothemel. Representing Time in Multimedia Systems. *International Conference on Multimedia Computing and Systems*, pages 538–543, May 1994.