Trabajo de Grado

Aplicando Métodos Formales a la Construcción de Aplicaciones de Hipermedia Colaborativas

Vanesa Mola y Wanda Marina Russo

Director: Lic. Luis Mariano Bibbó
Co-director: Dra. Claudia Pons

Agradecimientos

Queremos agradecer al Laboratorio de Investigación y Formación en Informática Avanzada (LIFIA), por habernos dado la oportunidad de pertenecer a un renombrado grupo de investigadores; a la gente de los proyectos en los que hemos participado: a Alicia Díaz y al grupo Biocom donde comenzamos nuestras tareas, y especialmente a Claudia Pons y a Luis Mariano Bibbó por la paciencia que nos tuvieron antes y durante el transcurso de esta tesis.

Además, queremos dar las gracias a las personas que nos ayudaron desinteresadamente en este trabajo, nuestros "consultores homologados": Pablo Pedemonte, Darío Silva, Federico Naso y Diego Cano.

Finalmente, no queremos olvidar a quienes compartieron con nosotras las horas de trabajo haciéndolas más amenas, la gente del LIFIA Multimedia: Inés Pederiva, Hernán Badenes, Sebastien Beignez, Darío Silva, Diego Cano, Gabriela Perez, Laura Ponisio, Bárbara Mercerat y Esteban de la Canal.

Wanda Marina Russo y Vanesa Mola

Quiero agradecer a mis padres, Eduardo y Argelia, por el constante apoyo y porque sin su ayuda esta carrera no habría sido posible; a mi abuela Carmen, por su atención desmedida a lo largo de todos estos años; no quiero olvidar al primer grupo de amigos de la facultad: Daniel, Agustina, Wanda, Diego, Hernán, Marcos, Chueko y Esteban; y a amigos más recientes: Inés y Darío. Finalmente quiero mencionar a Wanda, con quien fue un gusto compartir la realización de esta tesis.

Vanesa Mola

Agradezco principalmente a mis padres, por su apoyo, interés y ayuda siempre, no sólo durante el transcurso de mi carrera. Además, a todos mis amigos de la facultad, que hicieron de los últimos siete años una época especial.

A Pablo, mi ángel.

Wanda Marina Russo

La Plata, Septiembre de 2001

Este documento fue procesado usando el sistema de macros LaTeX 2ε, y el estilo para el lenguaje Object-Z desarrollado por Sebastian Rahtz.

Índice General

Índice de Figuras

Capítulo 1

Introducción

Muchas aplicaciones modernas como sistemas de documentación on-line, legislación, ingeniería de software, aplicaciones de negocios o sistemas para soportar toma de decisiones, se apoyan en las facilidades de los sistemas de hipermedia. Una de las ventajas de este tipo de sistemas es la organización de estructuras flexibles para representar la información, que pueden ampliarse y mantenerse fácilmente. Por otro lado, el mecanismo de navegación, que permite explorar la información, combina la potencia de cómputo con la participación del usuario que conduce la búsqueda. Esto permite realizar búsquedas exhaustivas sobre la información y es hoy un mecanismo naturalmente utilizado por una gran cantidad de usuarios de computadoras alrededor del mundo.

Por otro lado, los ambientes de hipermedia pueden ser utilizados por un conjunto de usuarios compartiendo el espacio de navegación y la información del sistema. Estos usuarios podrían querer realizar actividades en forma colaborativa, como editar un documento en forma conjunta, compartir una sesión de "brainstorming" o mantener una comunicación *on-line*. Esta combinación, sistemas de hipermedia con alguna funcionalidad de aplicaciones colaborativas, define un *ambiente de hipermedia colaborativo* [JCBZ99]. Dichos ambientes permiten a un conjunto de usuarios manipular y navegar sobre un conjunto de estructuras de hipermedia, de manera colaborativa. Éstos presentan tres principales diferencias con los ambientes *single-user* de hipermedia: el soporte para colaboración, comunicación y coordinación entre usuarios.

Los aspectos a tener en cuenta durante el diseño de un ambiente de hipermedia para soporte de colaboración son *group-awareness*, el acoplamiento del comportamiento de la aplicación y el contenido entre usuarios (*coupling*), el modelado de usuarios y sus roles en los escenarios colaborativos, la coordinación de la interacción de los usuarios (*floor control*), y la provisión de

canales de comunicación adicionales independientes del contenido [GF98].

Tanto el diseño conceptual como el de implementación de las aplicaciones de hipermedia colaborativas son tareas difíciles de llevar a cabo. A pesar de que han surgido herramientas y metodologías que facilitan el diseño de la implementación, todavía se dispone de poca asistencia para el diseño conceptual de este tipo de aplicaciones. OOHDM (Object Oriented Hypermedia Design Method) [Ros96] es una metodología de diseño que apunta a asistir en la creación de modelos para ambientes hipermediales *single-user*, sin embargo carece de soporte para características colaborativas. En [JCBZ99] se propone una extensión a OOHDM para soportar características colaborativas de una manera informal.

Un modelo se califica como informal cuando está expresado mediante lenguaje natural, figuras, tablas u otras notaciones. Por otra parte, nos referimos a modelos formales cuando la notación empleada es un formalismo, es decir, posee una sintaxis y semántica precisamente definidas.

El éxito de los lenguajes gráficos de modelado, como por ejemplo UML (Unified Modeling Language) [Gro99], se basa principalmente en el uso de construcciones gráficas que transmiten un significado intuitivo, resultando fáciles de entender y aplicar por parte de los usuarios. Sin embargo, la falta de precisión en la definición de su semántica puede originar diversos problemas de interpretación entre los desarrolladores, o inconsistencias entre los diferentes modelos de una misma aplicación. Por su parte, los lenguajes formales de modelado, tales como Z [DKRS91], poseen una sintaxis y semántica bien definidas, pero su uso en la industria es poco frecuente debido a la complejidad de sus formalismos matemáticos que son difíciles de entender y comunicar.

Volviendo al tema de aplicaciones de hipermedia colaborativas, es interesante observar cómo la aparición y el auge de la utilización de Internet en los últimos años ha permitido el crecimiento de las aplicaciones de hipermedia, dado que la mayoría de los documentos contenidos en ella son de naturaleza hipermedial. No resulta difícil imaginar que en un futuro no muy lejano la mayoría de las aplicaciones de hipermedia contenidas en Internet permitirán la colaboración entre los usuarios que compartan una aplicación, y brindarán la posibilidad de realizar tareas en forma conjunta, o sea, crecerá la utilización de las aplicaciones de hipermedia con características colaborativas. Actualmente ya existen algunos ejemplos en esta línea, como por ejemplo usuarios comentando libros o intercambiando opiniones sobre algún tema en foros de discusión.

En conclusión, el aporte de esta tesis consistirá en establecer una metodología de diseño para aplicaciones de hipermedia colaborativas, ya que en la actualidad este tipo de aplicaciones se construyen de manera artesanal. Por

otro lado, el enfoque planteado en este trabajo hace hincapié en la concepción y en el diseño, mientras que la mayoría de las investigaciones en el mundo sobre temas relacionados se focalizan sobre aspectos tecnológicos tales como compresión de datos multimediales, comunicaciones *on-line*, etc.

Finalmente, se proveerá una base formal a la metodología que permitirá realizar verificaciones de invariantes o reglas de buena formación en los modelos resultantes a partir de ella. Dicha base formal consistirá en la especificación de cada una de las clases definidas en el modelo de la metodología mediante el lenguaje formal Object-Z [RGG94].

1.1 Aplicaciones de Hipermedia

El concepto de hipermedia [Nie95] puede verse como una extensión al de hipertexto, ya que la única característica que diferencia a un concepto del otro es la inclusión de recursos multimediales, como video, sonido, etc; es decir, la información no sólo se circunscribe a texto específicamente. De ahora en más se utilizará el término hipermedia e hipertexto de manera análoga.

Hipermedia = **hiper**texto multi**medial**.

Un hipertexto es un texto organizado en módulos autocontenidos llamados *nodos*. Sea cual fuere la granularidad de los nodos, éstos pueden tener punteros a otras unidades de información, y a estos punteros se los llama *links*. Esta manera es más parecida a la manera que tienen las personas de enlazar pensamientos que la tradicional estructura lineal de los textos, ya que es más natural avanzar juntando ideas y asociándolas, y no siguiendo un hilo único y lineal.

La característica fundamental del hipertexto es que no suele tener linealidad; es posible llegar a un mismo nodo por varios caminos distintos. Por lo tanto, la estructura hipertextual forma una red de nodos y links a través de la cual los lectores avanzan en una actividad a la que comúnmente se llama *navegar*, algo más que simplemente "leer", para enfatizar que los usuarios deben determinar activamente el orden en el que visitan los nodos.

Una aplicación de hipermedia trabaja en colaboración con el usuario, quien tiene la inteligencia para entender el contenido semántico de los distintos nodos y es el que determina el próximo link a seguir.

No todas las aplicaciones son de naturaleza hipermedial. Para determinar cuándo una aplicación se adapta a las características de los sistemas de hipermedia, Shneiderman [Shn89] propuso lo que él llamó "las tres reglas de oro del hipertexto":

- Gran contenido de información organizado en numerosos fragmentos.
- Los fragmentos relacionados unos con otros.
- El usuario sólo necesita una pequeña fracción de información en un momento determinado.

Cualquier sistema de información podría incrementar su usabilidad y utilidad utilizando las características de las aplicaciones de hipermedia.

1.2 Aplicaciones Colaborativas

Un gran número de sistemas de software sólo contemplan la interacción entre el usuario y el sistema. Ya sea editando un documento o consultando una base de datos, el usuario interactúa solamente con la computadora. Inclusive sistemas diseñados para aplicaciones multiusuario, tales como sistemas de información para oficinas, proveen un soporte mínimo para la interacción *usuario-usuario*. Este tipo de soporte es claramente necesario, dado que una parte significativa de las actividades que desarrollan las personas se lleva a cabo en un contexto más bien grupal que individual. Para poder soportar esta interacción grupal será necesario abordar tres aspectos clave: *comunicación, colaboración* y *coordinación* [CGG91].

Una colaboración efectiva requiere que los usuarios compartan información. Desafortunadamente, los sistemas de información actuales, sistemas de bases de datos particularmente, apuntan a aislar a los usuarios unos de otros. Como ejemplo, se puede considerar dos diseñadores trabajando con una base de datos CAD (Computer Assisted Design). Un escenario ideal permitiría a los usuarios modificar diferentes partes del mismo objeto y a su vez que cada uno de ellos fuera notificado de los cambios que ocasionó el otro. La realidad es muy distinta a esto: cada uno de ellos deberá verificar el estado del objeto antes y después de sus modificaciones y notificarle al otro sobre los cambios que haya hecho. Muchas tareas requieren una granularidad aún más fina de compartimiento. Lo que se necesita son ambientes compartidos que ofrezcan de una manera natural un contexto grupal actualizado y notificaciones de las acciones de cada usuario cuando sea apropiado. Esto se conoce como *awareness*.

Según una definición de [CGG91]: *Las aplicaciones colaborativas, también conocidas como aplicaciones groupware, son sistemas basados en computadora que soportan grupos de usuarios asociados a una tarea común (o meta) y que proveen una interfase a un ambiente compartido.*

Los conceptos de *tarea común* y *ambiente compartido* son cruciales para la anterior definición. La misma excluye sistemas multiusuario, tales como sistemas de tiempo compartido, cuyos usuarios no comparten una tarea común.

En un entorno monousuario es importante notificar al usuario cuando determinadas restricciones se violan o cuando operaciones automáticas provocan *triggers* o alertas. La notificación o *awareness* se hace más importante aún en entornos multiusuario, ya que los usuarios deben saber cuándo otros usuarios efectúan cambios que impactan al trabajo común. Es por eso que las aplicaciones colaborativas deben hacer énfasis en el mecanismo de *awareness*, una forma de alertar y modificar la interfase de un usuario en respuesta a acciones llevadas a cabo por otro usuario en otra interfase.

Existen diferentes tipos de *awareness*, en [JCBZ99] se da una posible clasificación de los mismos. Los diferentes tipos tienen que ver con los usuarios de la aplicación, los objetos manipulados por ellos, las acciones que se llevan a cabo, capacidades de los distintos tipos de usuarios, etc.

1.3 Objetivos

De acuerdo a lo planteado anteriormente los objetivos del presente trabajo consisten en:

- Establecer una definición clara de *aplicación de hipermedia colaborativa.*
- Definir una metodología de diseño para aplicaciones de hipermedia colaborativas.
- Proveer a tal metodología de una base formal para evitar los problemas relacionados con los lenguajes de especificación gráficos.
- Mostrar la utilización de la metodología definida mediante ejemplos.

En el capítulo 2 se dará una breve introducción a la metodología de diseño de aplicaciones de hipermedia OOHDM [Ros96] a partir de la cual se definió la extensión para soporte de características colaborativas, principal aporte de este trabajo de grado. Dicha extensión, denominada CHDM (Collaborative Hypermedia Design Method), se describe en el capítulo 3 en el cual se provee una definición de *aplicación de hipermedia colaborativa.* CHDM reusa las etapas descriptas en OOHDM y agrega una más, la etapa del modelo colaborativo. Dichas etapas (o modelos) se encuentran definidas de manera informal en [Ros96], utilizando simplemente el lenguaje natural y algunos diagramas

de clase. Es por eso que se vio la necesidad de clarificar los conceptos que se plantean utilizando una notación formal. Primero se emplearon diagramas de UML para representar los aspectos estructurales de cada una de las etapas y posteriormente, en la etapa colaborativa propuesta, se utilizaron diagramas de secuencia para modelar aspectos dinámicos de comportamiento.

Luego, en el capítulo 4, se muestra una especificación formal, mediante el lenguaje de especificación formal orientado a objetos Object-Z [Gri97, RGG94, KC99], para cada una de las clases de las etapas de CHDM como formalismo para detallar cuestiones más finas que no se deducían claramente de la especificación hecha en UML, y a modo de base formal subyacente para tal metodología. Además, se muestra con ejemplos las verificaciones formales que pueden realizarse gracias a la formalización.

Para finalizar, en el capítulo 5 se muestra la aplicación de CHDM a dos ejemplos: una propuesta de aplicación colaborativa a partir de la aplicación de hipermedia del Museo de Portinari [TS] diseñada previamente con OOHDM y un posible diseño para la aplicación colaborativa DOLPHIN [SHH94], desarrollada por el laboratorio GMD-IPSI [GMD].

Además, se proveen los apéndices A y B sobre la notación básica de UML y Object-Z necesarias para la comprensión de los diagramas incluidos en el presente trabajo.

Capítulo 2

OOHDM

En este capítulo se describirá brevemente la metodología de diseño de aplicaciones de hipermedia orientada a objetos OOHDM. El lector interesado en los detalles de esta metodología podrá remitirse a [Ros96].

OOHDM es una metodología de diseño para construir aplicaciones de hipermedia basada en modelos. Fue definida a partir de la metodología HDM [FSP93] con un enfoque orientado a objetos. Considera el proceso de desarrollo de aplicaciones de hipermedia como un proceso de cuatro actividades, desempeñadas en una combinación de estilos iterativos e incrementales de desarrollo, en el cual en cada etapa un modelo es construido o enriquecido. Estas actividades son: diseño del modelo conceptual, diseño del modelo navegacional, diseño de la interfase abstracta e implementación. De cada una de las actividades se puede pasar a la siguiente (evolución natural del desarrollo del proyecto) como así también se puede volver atrás en forma de retroalimentación.

Durante el diseño del modelo conceptual, se construye un modelo del dominio de la aplicación utilizando principios bien conocidos de la orientación a objetos aumentados con algunas primitivas tales como perspectivas de atributos y subsistemas. Las clases conceptuales pueden construirse utilizando jerarquías de agregación y de generalización/especialización. En este punto no es menester preocuparse por los tipos de usuario o tareas que éstos deban realizar en el sistema. El resultado de esta etapa es un esquema de clases y objetos construido a partir de subsistemas, clases y relaciones.

Una de las características de las aplicaciones de hipermedia es la noción de navegación. En OOHDM, una aplicación es vista como una visión navegacional del modelo conceptual. Esa visión es construida durante el diseño del modelo navegacional teniendo en cuenta los tipos de usuarios a los que está destinada la aplicación y los tipos de tareas que deberán desempeñar utilizándolo. Diferentes esquemas navegacionales pueden ser definidos para

un mismo esquema conceptual, expresando, de esta forma, diferentes visiones del mismo dominio. La estructura navegacional de una aplicación de hipermedia es descripta definiendo las clases navegacionales que reflejen la visión elegida del dominio de la aplicación. Esta estructura es definida en términos de contextos navegacionales, que son inducidos (de diferentes maneras, dependiendo del tipo de contexto) a partir de clases navegacionales tales como Nodos o Links.

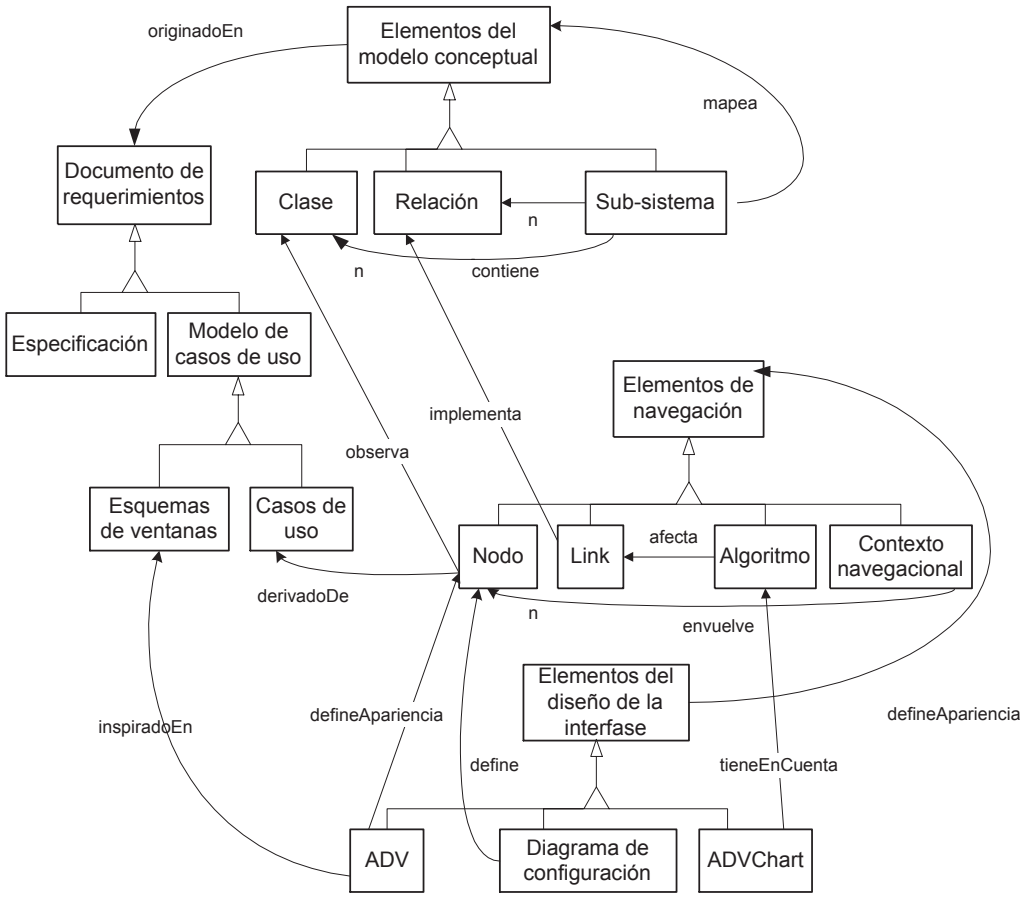

Figura 2.1: Modelo de OOHDM

Durante el diseño de la interfase abstracta se construye un modelo de la interfase, que especifica qué objetos de la interfase serán vistos por el usuario y la forma que tomarán los diferentes objetos navegacionales, qué objetos activarán la navegación, cómo serán sincronizados los objetos de multimedia y qué transformaciones ocurrirán en la interfase.

La clara separación entre la interfase abstracta y el modelo navegacional permite construir interfases diferentes para un mismo modelo navegacional adecuándose a las necesidades o preferencias del usuario como así también a las diferentes tecnologías para la construcción de interfases.

Finalmente se produce un sistema real de hipermedia mapeando los modelos navegacional y de interfase a un ambiente de implementación elegido.

En la figura 2.1 se puede observar una definición gráfica de las etapas de OOHDM junto con las relaciones existentes entre ellas. Dicho diagrama fue extraído de [Ros96]. Allí se describen tres jerarquías principales: elementos de los modelos conceptual, navegacional y de interfase, y una adicional, documento de requerimientos (que incluye la especificación y los modelos de casos de uso). Se omitió la etapa de implementación por razones de simplicidad del diagrama.

En la sección 2.1 se describirá la etapa del modelo conceptual, en la sección 2.2 la etapa del modelo navegacional, luego la etapa del modelo de interfase abstracta en la sección 2.3. La etapa de implementación se describirá en la sección 2.4 y finalmente en la sección 2.5 se comentarán las contribuciones de OOHDM al diseño de aplicaciones de hipermedia.

2.1 Modelo Conceptual

En OOHDM el Modelo Conceptual está formado por clases, relaciones y subsistemas. Las clases son descriptas como de costumbre en modelos orientados a objetos, salvo por el hecho de que los atributos pueden ser multitipados, representando diferentes perspectivas de una misma entidad real. Se utiliza una notación similar a OMT (Rumbaugh) enriquecida con información de subsistemas. Además se utilizan cartas de Clases y Relaciones similares a las CRCs para especificar con mayor detalle cada una de las clases y relaciones para asistir en la documentación.

El modelo conceptual equivale a la etapa de Análisis en las metodologías orientadas a objetos.

2.1.1 Primitivas de Modelado

Como fue mencionado previamente, se tienen clases, relaciones y subsistemas. Las clases pueden, a su vez, relacionarse con subsistemas. Un subsistema puede ser autocontenido (con un solo punto de entrada y salida a modo de servidor de información). Los puntos de entrada de un subsistema son clases que pueden ser accedidas a partir de otras clases o subsistemas.

Clases y Relaciones

Durante el Modelo Navegacional las clases serán mapeadas a nodos, mientras que las relaciones serán convertidas en links. La decisión de representar las relaciones como atributos, como métodos o como una combinación de ambos ha sido ampliamente discutida en la comunidad de orientación a objetos. En realidad, los enfoques actuales de orientación a objetos tratan a las relaciones como "ciudadanos de primera clase" relegando la discusión a las etapas de implementación del proyecto.

Los atributos no deben representar entidades conceptuales complejas, esto debe ser representado como una relación. En el caso en que no se esté interesado en ambos como conceptos separados sí podría representarse uno como atributo del otro.

Las relaciones también pueden tener atributos (y eventualmente comportamiento) y se debe indicar la cardinalidad de las mismas.

Atributos, Tipos y Perspectivas

Los atributos son tipados y representan propiedades intrínsecas a los objetos. El tipo o clase de un atributo representará, en la aplicación de hipermedia final, una relación implícita, o la apariencia visual de este atributo. Cada apariencia posible es denominada "perspectiva del atributo". En el caso de existir múltiples perspectivas se utilizan "[...]" y en caso de existir una default será marcada con un $^{+}$. La perspectiva default debe estar presente en todas las instancias, las demás pueden no ser implementadas.

La especificación del tipo de un atributo como una clase permite expresar perspectivas de manera más genérica. Es posible construir una jerarquía de la clase A con subclases y decir que un atributo tiene clase A es decir que puede tomar uno o varios tipos de cualquiera de las subclases también.

Mecanismos de Abstracción

Se proveen tres mecanismos de abstracción: agregación, generalización/especialización y un concepto de paquetes o subsistemas. El primero es útil en la descripción de clases complejas como agregaciones de clases más simples. El segundo es utilizado en la construcción de jerarquías de clases y en el uso de la herencia. Los subsistemas son un mecanismo de empaquetamiento para abstracción de modelos de dominio complejos.

Las relaciones *parte-de* son descriptas a través de relaciones de agregación. Existen relaciones implícitas entre un objeto y sus partes (y viceversa) y entre las partes. En HDM [FSP93] dichas relaciones son llamadas links estructurales.

Descripciones Basadas en Instancias

Un esquema conceptual permite la descripción de una familia de aplicaciones de hipermedia sobre determinado dominio. Las abstracciones de generalización/especialización y de composición, además del mecanismo de herencia, sirven de base para el reuso de componentes de hipermedia.

Un esquema de instancias (i.e. esquema de instancias en una aplicación específica) generalmente refleja la estructura del esquema de clases pues los objetos son creados de acuerdo a las definiciones de sus clases.

Sin embargo, en las aplicaciones de hipermedia puede ocurrir que ciertos objetos exhiban características excepcionales que no fueron consideradas en la definición de su clase. En muchos casos esto se puede resolver creando subclases más especializadas, pero tal solución no tiene sentido si el número de instancias excepcionales es pequeño o existen muchos tipos de instancias excepcionales pues se complica el esquema de clases.

En OOHDM se propone un esquema de instancias para expresar esta situación. El esquema de instancias muestra las instancias excepcionales, i.e. aquellas que no corresponden exactamente a la definición de los esquemas de clases. En este esquema se pueden agregar atributos (o partes) a un objeto, cambiar el tipo de los atributos existentes, refinar las relaciones, mostrar valores posibles, etc.

La definición de un esquema de instancias está relacionada con el proceso de prototipación.

Resumen de los Resultados del Diseño Conceptual

A partir de la definición de un diseño conceptual se obtendrán:

- Un esquema conceptual (en realidad un conjunto, cada uno perteneciendo a un subsistema)

- Un conjunto de definiciones de subsistemas, clases, objetos y relaciones (utilizando tarjetas de subsistemas, clases, objetos y relaciones)

Las tarjetas incluyen información sobre los objetos documentados tal como referencias "hacia atrás" que permiten responder a preguntas como: *¿de dónde viene este objeto?* (una entidad del mundo real, o una relación, por ejemplo), o referencias "hacia adelante", importantes para analizar el impacto de los cambios en el modelo durante la evolución del sistema (*¿en qué clases puede influir el cambio?*).

2.2 Modelo Navegacional

A partir del modelo conceptual, que hace hincapié en los objetos y relaciones del dominio de aplicación, puede derivarse la construcción de diferentes vistas navegacionales, teniendo en cuenta el perfil de distintos usuarios. O sea, se pueden definir varios modelos navegacionales distintos para un mismo modelo conceptual.

En términos de arquitectura orientada a objetos, las relaciones entre los nodos y los objetos conceptuales, y entre los links y las relaciones del diseño conceptual se expresan como instancias del patrón de diseño *Observer* (ver [GHJV94, p.293]).

2.2.1 Clases Navegacionales Básicas

Nodos

Los nodos representan vistas lógicas de las clases definidas en el diseño conceptual. Se describen a través de un conjunto de atributos y un conjunto de métodos que implementan comportamiento y se organizan en jerarquías *parte-de* y *es-un*. Los atributos pueden ser de distintos tipos: aquellos que contienen información que será mostrada en la aplicación y los anchors, siendo *Anchor* otra clase básica en OOHDM. Los nodos pueden ser atómicos o compuestos y se definen a través de atributos y anchors hacia links en el caso de los atómicos, y como un conjunto de nodos componentes en el caso de ser compuestos.

En la figura 2.2 se muestra la jerarquía de la clase Nodo. Los nodos pueden ser visitados en diferentes contextos y sus atributos y anchors pueden cambiar de un contexto a otro. Por ejemplo, cuando se accede a las obras de arte de un mismo tema se tiene un anchor apuntando a un índice de temas. Esto se logra mediante la "decoración" del nodo (ver patrón de diseño orientado a objetos *Decorator* [GHJV94, p.175]) y la definición de clases *EnContexto*, como se verá mas adelante.

Links

Los links, por su parte, pueden ser derivados de las relaciones en la etapa del diseño conceptual (links de aplicación), generados implícitamente a partir de la definición de contextos navegacionales (links contextuales) o bien agregados arbitrariamente por el diseñador (links arbitrarios). Los links efectúan la unión entre objetos navegacionales (nodos) y pueden tener cardinalidad de uno a uno o de uno a muchos. El resultado de pasar a través de un link se expresa en la definición de la semántica navegacional, como resultado del

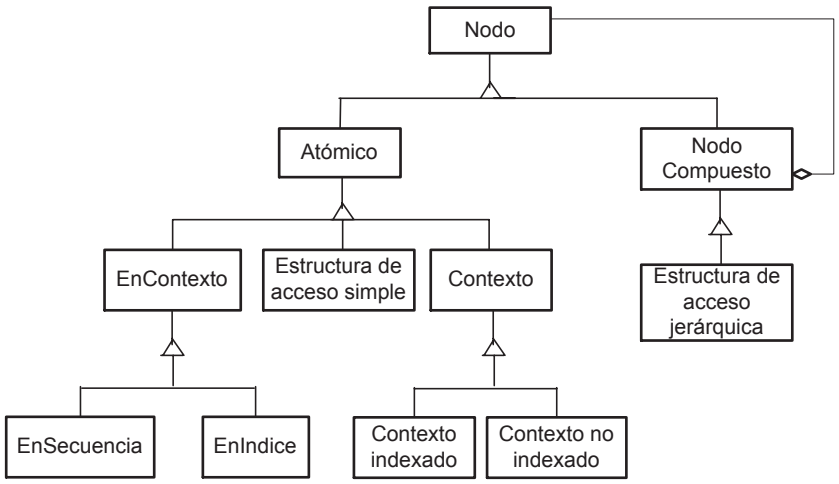

Figura 2.2: Jerarquía de Nodo

comportamiento del link. Las estructuras de acceso, tales como índices o rutas guiadas, también se definen como clases y presentan modos alternativos para la navegación en una aplicación de hipermedia.

Contextos

Los contextos navegacionales son un conjunto de nodos, links y otros contextos navegacionales (anidados) que ayudan en la organización de los objetos navegacionales y forman un espacio de navegación consistente que evita que el usuario se pierda. Un contexto navegacional puede ser también un camino predefinido entre los nodos que lo componen. Los contextos navegacionales definen un conjunto de clases "decoradoras" que adaptan cada nodo a un contexto agregando al nodo las estructuras (anchors, atributos, etc) necesarias para ese contexto. Éstos son los nodos *EnContexto*, que permiten que un nodo tenga apariencia distinta y muestre diferentes características dependiendo del contexto en el que es visitado.

Los contextos navegacionales ejercen un papel similar al de las clases en las Bases de Datos Orientadas a Objetos. Tales contextos son un conjunto de instancias relacionadas, como por ejemplo, todas las obras de arte, las obras sobre un determinado tema, todas las obras de un mismo pintor, un conjunto aleatorio de obras en una ruta guiada, etc.

En la figura 2.3 se muestra la notación de OOHDM para la definición de contextos. Dicha notación permite la especificación de contextos, familias de contextos, estructuras de acceso, etc.

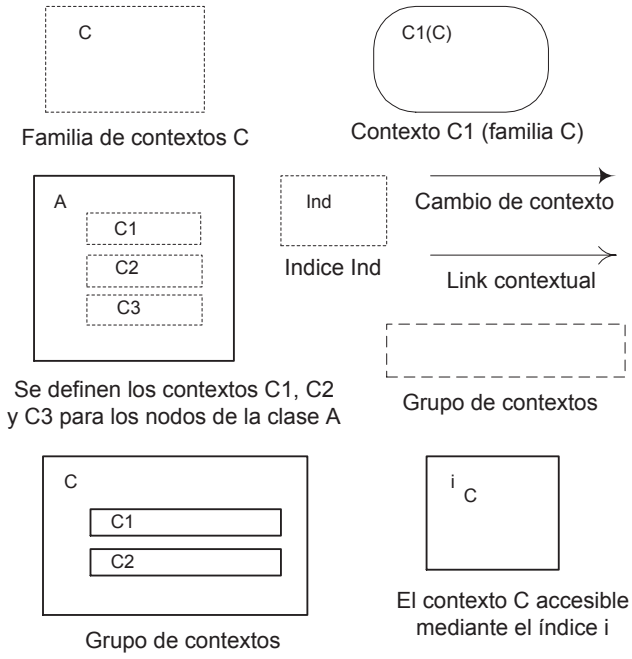

Figura 2.3: Notación para la Definición de Contextos

Las clases y contextos navegacionales definen la estructura estática de la aplicación de hipermedia. También es necesario especificar los aspectos dinámicos de la navegación. En OOHDM, esto se hace a través de los diagramas de navegación, un modelo basado en máquinas de estados, en el cual se muestran las transformaciones de estados en el espacio navegacional.

2.3 Modelo de la Interfase Abstracta

OOHDM propone el diseño formal de la interfase previo a la implementación, de manera de maximizar la independencia de diálogo y el reuso a gran escala de los componentes de interfase. Mientras que el modelo navegacional define *cuál* es la estructura de la navegación de la aplicación, la tarea del diseño de la interfase abstracta es definir *cómo* se presentará al usuario tal estructura navegacional. Los elementos de diseño utilizados en esta etapa son los ADVs (Abstract Data Views), que permiten especificar un modelo de Interfase Abstracta. Los ADVs son objetos en el sentido en que poseen un estado y una interfase de mensajes. A su vez los ADVs representan ADOs (Abstract Data Objects) de la aplicación.

En OOHDM los elementos navegacionales funcionan como ADOs a los

que se les definirán ADVs para especificar su apariencia frente al usuario.

En la figura 2.4 se muestra un ejemplo de definición del ADV para el nodo *Persona*, el cual incluye atributos tales como nombre, descripción, biografía y *email*. Este último es un anchor.

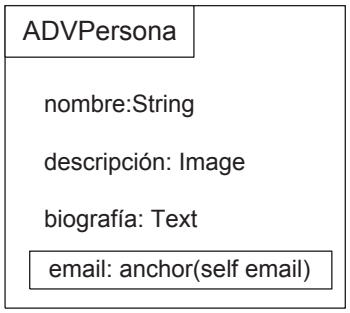

Figura 2.4: Ejemplo de ADV para el nodo Pintura

Los ADVs, a su vez, se organizan en ADVCharts y Diagramas de Configuración para especificar relaciones estáticas y dinámicas entre los ADVs y los ADOs. Los ADVCharts son semejantes a los diagramas de navegación.

Los pasos para construir una especificación de interfase de hipermedia son los siguientes:

- definir la estructura general de la interfase de la aplicación.
- definir ADVs para los nodos, índices, etc.
- definir para cada nodo, objetos de interfase adecuados para sus atributos, anchors, etc.
- definir ADVs para clases de contextos y nodos *EnContexto*.
- especificar diagramas de configuración, mostrando las relaciones estáticas entre los componentes de un ADV.
- especificar para cada ADV significativo un ADVChart mostrando la dinámica de la aplicación de hipermedia.

2.4 Etapa de Implementación

La etapa de Implementación, básicamente, es la responsable de la traducción del Modelo Navegacional y el de Interfase a un ambiente de Implementación.

Se refiere a definir los objetos de interfase de acuerdo con la especificación del modelo de Interfase Abstracta, implementar las transformaciones como fueron definidas en los ADVCharts y proveer soporte para la navegación a través de redes hipermediales.

Existen muchas maneras de implementar un modelo y dependen fuertemente de las herramientas que se utilicen para tal fin. Por tal razón, no se ha ahondado en la definición de esta etapa.

Si la aplicación fuera implementada para la Web, existe una herramienta que permite automatizar la creación de la aplicación diseñada con OOHDM. Tal herramienta es OOHDMWeb [Sch], la cual crea una aplicación Web hipermedial a partir de un modelo navegacional mapeado a tablas de Microsoft Access.

2.5 Contribuciones Importantes de OOHDM

Según [Ros96], las principales contribuciones de la metodología de OOHDM son:

- Una separación clara entre la etapa de análisis (modelo conceptual) y diseño (modelos navegacional y de interfase) dada la diferencia de primitivas de modelado para cada etapa.
- Una visión completa de los proyectos de las aplicaciones de hipermedia, teniendo en cuenta sus diversas actividades (como el diseño conceptual, de navegación y de interfase).
- Un conjunto uniforme de formalismos orientados a objetos y mecanismos de abstracción que soportan la construcción de aplicaciones de hipermedia.
- Un conjunto de patrones de arquitectura de hipermedia para ser utilizados durante el ciclo de desarrollo.
- Un modelo simple de documentación para ayudar durante la evolución de la aplicación.

Capítulo 3

CHDM

En la actualidad, es posible encontrar diversas aplicaciones de naturaleza hipermedial (gran cantidad de información organizada en nodos navegables a través de links), tales como enciclopedias, cursos, sistemas de documentación *on-line*, etc. Muchas de las características de las aplicaciones de hipermedia también pueden incluirse en cualquier sistema de información para incrementar su usabilidad y utilidad. Una de las ventajas de este tipo de sistemas es la organización de estructuras flexibles para representar la información, que pueden ampliarse y mantenerse fácilmente. Además, el mecanismo de navegación, que permite explorar la información, combina la potencia de cómputo con la participación del usuario que conduce la búsqueda. Esto permite realizar búsquedas exhaustivas sobre la información y es hoy un mecanismo naturalmente utilizado por una gran cantidad de usuarios de computadoras alrededor del mundo.

También es común encontrar aplicaciones de índole colaborativa (sistemas basados en computadora que proveen entornos compartidos en los cuales un número de usuarios se comunica e interactúa para desarrollar una tarea común, ya sea sincrónica o asincrónicamente). Se pueden mencionar las herramientas de *e-mail*, ICQ, Net Meeting, como claros exponentes de tales aplicaciones.

Por otro lado, es interesante observar cómo la aparición y el auge de la utilización de Internet en los últimos años ha permitido el crecimiento de las aplicaciones de hipermedia, dado que la mayoría de los documentos contenidos en Internet son de naturaleza hipermedial. No resulta difícil imaginar que en un futuro no muy lejano las aplicaciones de hipermedia contenidas en Internet permitirán la colaboración entre los usuarios que compartan una aplicación, y brindarán la posibilidad de realizar tareas en forma conjunta, o sea, crecerá la utilización de las aplicaciones de hipermedia con características colaborativas.

Uno de los principales objetivos de esta tesis, fue plantear una definición concreta de las aplicaciones de hipermedia colaborativas:

Una aplicación de hipermedia colaborativa provee, al igual que una aplicación de hipermedia, gran cantidad de información organizada en nodos navegables a través de links y además provee una interfase a un ambiente compartido, permitiendo que grupos de usuarios puedan "encontrarse" en nodos específicos para llevar a cabo una tarea colaborativa o efectuar una navegación en forma grupal.

En el laboratorio GMD-IPSI [GMD], por ejemplo, se investigan las aplicaciones de hipermedia colaborativas y se han desarrollado numerosas herramientas que asisten a la colaboración de un grupo de usuarios; una de ellas es SEPIA [HHL+92, SHH+92], un ambiente que permite la construcción de documentos de hipermedia por un grupo de autores distribuidos en diferentes computadoras y hasta en diferentes ciudades. Los usuarios participantes pueden establecer sesiones sincrónicas, donde todos comparten la misma "vista" del documento (WYSIWIS - What You See Is What I See), o trabajar en modo asincrónico donde los cambios hechos por un usuario se ven reflejados en las vistas de los demás. Su sucesora, DOLPHIN [SHH94], se especializa en reuniones de usuarios en ambientes virtuales para trabajo cooperativo (CSCW - Computer Supported Cooperative Work) y está basada en un modelo de datos hipermedial. En DOLPHIN los usuarios pueden trabajar en un ambiente privado, sin interacción con los demás, o en ambientes públicos de manera grupal. Además, se provee información de awareness sobre el resto de los usuarios que se encuentran en el mismo ambiente y se permite la participación de usuarios de manera remota mediante video-conferencias.

Sin embargo, no se cuenta con una metodología de diseño que asista al desarrollador de aplicaciones de hipermedia colaborativas en su trabajo. Tales aplicaciones se desarrollan de manera intuitiva, lo que hace que sean difíciles de ampliar o mantener, o que contengan errores de diseño.

El presente trabajo de grado pretende dejar un punto de partida claro para diseñar aplicaciones de hipermedia colaborativas. Para ello se propone una extensión a OOHDM dado que no soporta el diseño de ese tipo de aplicaciones. Dicha extensión, denominada CHDM (Collaborative Hypermedia Design Method), permite modelar características de índole colaborativa en aplicaciones de hipermedia agregando una nueva etapa llamada *colaborativa*.

Los aportes de este capítulo consisten en: plasmar en un diagrama de clases UML un diseño orientado a objetos para las etapas conceptual y navegacional definidas en OOHDM (en la manera en que son reutilizadas en CHDM) y luego, en base a dichas etapas, definir y elaborar un diseño propio

para la etapa colaborativa.

Las etapas de diseño de interfase abstracta e implementación definidas en OOHDM quedan fuera del alcance de este trabajo puesto que dependen fuertemente de la arquitectura y herramientas que se empleen.

3.1 Características Colaborativas

En esta sección se definirán y explicarán brevemente las características que se presentan en las aplicaciones colaborativas. Las mismas son: *awareness*, $floor - control$, usuarios y roles, y sesiones de colaboración entre usuarios. Dichas características deben ser tenidas en cuenta a la hora de diseñar e implementar toda aplicación colaborativa.

3.1.1 Awareness

La dificultad inherente de la comunicación mediante una computadora es que tal medio reduce o elimina la visibilidad salvo que se tomen medidas para aumentarla apropiadamente. Históricamente, los sistemas y aplicaciones multiusuario trataban de aislar a los usuarios y sus trabajos de los demás, para dar la ilusión de que el sistema tenía sólo un usuario. Pero el trabajo colaborativo presenta situaciones en que los usuarios deben interactuar directamente, no sólo con la computadora sino entre ellos. En tal caso, la estrategia de aislamiento es lo contrario a lo necesario. Las interacciones entre usuarios se pueden clasificar como muestra la tabla siguiente.

	Mismo tiempo	**Diferente tiempo**
Mismo lugar	cara a cara	asincrónica
Diferente lugar	sincrónica distribuida	asincrónica distribuida

El rol del sistema es proveer una interfase al ambiente compartido, puesto que estar en el mismo lugar no es posible cuando los participantes están geográficamente dispersos. Esto implica que los participantes tengan conocimiento de los demás, del ambiente y de las tareas que se encuentran desarrollando; a esto se lo llama *awareness*.

En [JCBZ99] se describen los distintos tipos de *awareness* que se pueden encontrar en un sistema colaborativo, tales como: *awareness* de presencia, ubicación, actividad, objeto, cambios, etc.

3.1.2 Usuarios y Roles

En los sistemas multiusuario, generalmente los usuarios tienen diferentes permisos o posibilidades de acceso a los objetos del sistema. Por ejemplo, cada usuario puede acceder a sus archivos y no a los de los demás.

Por otro lado, pueden existir grupos de usuarios con características comunes con respecto a su relación con el sistema y con el resto de los usuarios, para eso se definen roles que abstraen las características comunes a tales grupos. Por ejemplo, un usuario con rol de *administrador* tiene permiso para realizar funciones que el resto de los usuarios no puede hacer tal como crear un nuevo usuario del sistema.

En los sistemas colaborativos es necesario modelar usuarios y roles puesto que éstos pueden intervenir en tareas conjuntas con diferentes permisos y responsabilidades.

3.1.3 Floor Control

Floor control en aplicaciones colaborativas sincrónicas es el problema de cómo, cuándo, y por qué los participantes interactúan en un ambiente compartido de computación trabajando de manera simultánea en tareas comunes [JAB96]. El termino *floor* se utiliza para denominar al "turno de hablar".

La mayor dificultad en los sistemas centralizados radica en manejar las actividades de los diferentes usuarios. Una manera es permitir que sólo un usuario a la vez provea input a la aplicación. Las arquitecturas distribuidas presentan un problema diferente, si cada usuario está autorizado a controlar su instancia de la aplicación separadamente se debe hacer algo para sincronizar las diferentes instancias. De nuevo se debe sincronizar el input de los diferentes usuarios. Esto se puede efectuar de diferentes maneras: la aplicación podría permitir input de un solo usuario a la vez (como en los sistemas centralizados), o se podría asegurar que todas las instancias tengan los mismos input mediante broadcasting del input de un usuario a todas las demás instancias de la aplicación. Pero este enfoque introduce un nuevo problema: las instancias separadas podrían recibir input en diferentes órdenes y no cualquier programa garantiza comportarse consistentemente bajo tales circunstancias.

El caso simple de floor control es el de pasar un floor global simple, es decir, se permite a un usuario a la vez tener el control y el control es pasado explícitamente entre usuarios. Pero un solo control global no es la única posibilidad. Los usuarios podrían querer tener diferentes roles y así diferentes vistas del mismo ambiente. Entonces puede ser útil menores granularidades de floor control.

3.1.4 Sesiones de Colaboración

En los sistemas colaborativos, cuando un usuario colabora con otro se establece una sesión de colaboración. Tales sesiones pueden tener un alto nivel de acoplamiento, como cuando un grupo de usuarios se encuentra utilizando un editor colaborativo, donde todos ven reflejados los cambios que provocan los demás sobre el documento a editar; o un bajo nivel de acoplamiento, donde todos los usuarios trabajan en forma separada pero cada uno ve qué está haciendo el resto.

Cuanto más acoplada sea la sesión de colaboración, menos esfuerzo es necesario para la implementación del mecanismo de awareness. Cuando los usuarios trabajan de manera fuertemente acoplada todos "ven" la misma vista de la aplicación, por lo tanto no es necesario proveer información adicional para que estén al tanto de las actividades del resto de los usuarios.

3.2 OOHDM como Punto de Partida

Como se dijo en la sección anterior, los puntos que las aplicaciones colaborativas deben tener en cuenta son la provisión de *awareness* (notificación al usuario de eventos que ocurren como consecuencia de acciones de otros usuarios o del sistema mismo), roles de usuario, control de acoplamiento, sesiones de colaboración y *floor-control*.

En esta sección se verá que OOHDM no provee soporte para la mayoría las características anteriormente mencionadas. Sin embargo, provee una metodología para el diseño de aplicaciones de hipermedia que cubre todas las etapas de desarrollo. Por tales razones se decidió realizar una extensión a OOHDM para tener en cuenta las características de las aplicaciones colaborativas.

En lo que sigue se verán las limitaciones de OOHDM a la hora de diseñar una aplicación de hipermedia colaborativa y la forma en que la extensión soluciona tales carencias.

3.2.1 Limitaciones de OOHDM

La metodología de diseño OOHDM (capítulo 2) no fue pensada originalmente para soportar el diseño de aplicaciones de hipermedia colaborativas. A continuación se listan algunas razones por las cuales OOHDM no permite modelar dichas aplicaciones:

1. No permite modelar a los usuarios en particular y las relaciones entre ellos.

2. No permite vincular herramientas de edición o colaboración a la estructura de navegación.

3. Modela principalmente aspectos estructurales y no dinámicos en el diseño de las estructuras navegables.

A continuación se verán más en detalle cada uno de los puntos anteriores:
1. *No permite modelar a los usuarios en particular y las relaciones entre ellos.* OOHDM contempla la posibilidad de tener diferentes tipos de usuarios que utilicen la aplicación. Para esto es necesario definir un modelo navegacional para cada uno de ellos, dado que tendrán acceso a diferentes partes del sistema y podrán realizar diferentes tipos de tareas. Pero no permite representar a los usuarios en particular, junto con los roles que pueden jugar en la aplicación, ni las relaciones entre ellos, o la posibilidad de comunicación e interacción. Además, al no tener en cuenta a los usuarios en particular, tampoco soporta el control de concurrencia entre diferentes instancias de la aplicación o características colaborativas tales como *awareness*.

2. *No permite vincular herramientas de edición o colaboración a la estructura de navegación.* OOHDM sólo modela aplicaciones de hipermedia como un conjunto de nodos en los cuales se puede navegar, pero no permite agregar funcionalidad a los nodos, tal como herramientas de edición o colaboración. Dichas herramientas son indispensables en una aplicación colaborativa. Por ejemplo, un grupo de usuarios colaborando mediante el uso de una aplicación colaborativa necesita alguna herramienta de comunicación, tal como *Net Meeting*, herramientas de *e-mail*, etc.

3. *Modela principalmente aspectos estructurales y no dinámicos en el diseño de las estructuras navegables.* OOHDM modela los aspectos estructurales de las aplicaciones de hipermedia, los objetos que las componen (nodos, links, anchors), y el espacio navegacional que conforman; pero en las aplicaciones de hipermedia colaborativas se necesita también modelar aspectos de comportamiento puesto que la interacción con el sistema no sólo se circunscribe a un usuario explorando un espacio de navegación, sino también a usuarios interactuando entre sí y con el sistema. Es decir, OOHDM permite especificar información estática de la aplicación, pero sus primitivas de modelado no incluyen, por ejemplo, diagramas de secuencia o interacción para especificar aspectos dinámicos. Algunos aspectos dinámicos que se deberían poder modelar son: navegación de manera acoplada de grupos de usuarios, representación del *awareness* entre usuarios, sesiones de colaboración, usuarios y roles.

3.2.2 Nueva Etapa: el Modelo Colaborativo

CHDM reusa las etapas de OOHDM del modelo conceptual, modelo navegacional, diseño de la interfase abstracta e implementación vistas en el capítulo 2; pero agrega una más, el modelo colaborativo, dedicada a agregar los aspectos de aplicaciones colaborativas tales como identificación y manejo de los usuarios de la aplicación y sus roles, sesiones de colaboración entre ellos junto con la manera en que interactúan y colaboran, *awareness*, acoplamiento y herramientas colaborativas sincrónicas y asincrónicas. *Floor-control* no será tenido en cuenta en este trabajo dado que atañe principalmente a la implementación.

La etapa de diseño colaborativo se especifica posteriormente a la del diseño navegacional y consiste en identificar los nodos de la aplicación a los que se les agregará alguna de estas características y modelar nodos colaborativos para ellos.

Como resultado de la nueva etapa, se obtiene un modelo de objetos concreto para especificar de manera estructural las características colaborativas de la aplicación, y diagramas de secuencia que permiten especificar la parte dinámica de la aplicación, es decir, el comportamiento de dichos objetos en escenarios colaborativos puntuales y en qué manera colaboran.

En lo que respecta al usuario de CHDM, un diseñador de aplicaciones de hipermedia colaborativas, a la hora de construir el modelo de una aplicación de hipermedia colaborativa solamente necesita instanciar cada uno de los modelos de CHDM que se mostrarán a continuación. Dicho proceso consiste de los siguientes pasos:

- Construir el modelo conceptual identificando los objetos y relaciones del dominio de la aplicación.

- Construir el modelo navegacional con la estructura navegacional de la aplicación.

- Identificar las características colaborativas que se desean agregar a la aplicación y diseñar nodos colaborativos para cada nodo al que se quiera agregar dichas características.

- Instanciar las metaclases provistas por el modelo colaborativo de acuerdo a las necesidades de la aplicación.

Tanto los diagramas estáticos (de objetos) como los dinámicos (de secuencia) surgen de la instanciación de los modelos propuestos en esta tesis. El diseñador, además, puede subclasificar las clases que necesite o puede agregar

los diagramas de secuencia que considere adecuados para la mejor comprensión del modelo por parte de su equipo de desarrollo.

A continuación se presentará el modelo de la metodología CHDM en sus diferentes etapas. Nótese que con estos diagramas se ve conformado un metamodelo de las aplicaciones de hipermedia colaborativas. O sea, representan un modelo que realiza una abstracción de las características presentes en todo modelo de aplicación de hipermedia colaborativa. Por lo tanto, CHDM es lo suficientemente general para soportar el diseño de cualquier aplicación de este tipo.

3.3 Metamodelos CHDM

Antes de abordar el tema de los metamodelos planteados para CHDM, se hará una breve descripción respecto a la arquitectura conceptual de tres niveles utilizada.

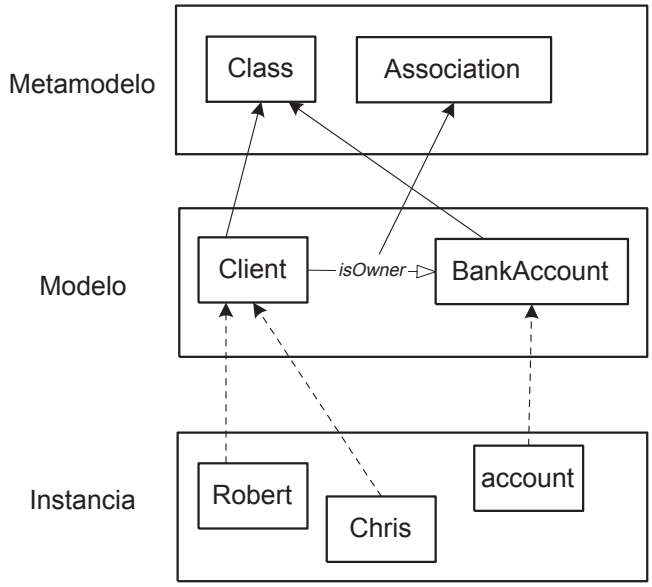

Figura 3.1: Tres niveles: Instancia, Modelo y Metamodelo

En la figura 3.1 se describen tres niveles de abstracción. El nivel de abstracción crece a medida que se asciende en el diagrama. El nivel inferior representa el nivel de instancias, donde se pueden encontrar objetos concretos de un dominio particular, como por ejemplo el cliente *Robert*, la cuenta bancaria *account*, etc. El siguiente nivel, es el nivel de los modelos. Las entidades participantes de este nivel son elementos que modelan un dominio de

aplicación en particular, como por ejemplo las clases *Client*, *BankAccount*, y la relación *isOwner*. Ascendiendo un nivel más de abstracción se encuentra el nivel de metamodelos, en donde las entidades que habitan son entidades comunes a todos los modelos, o, mejor dicho, entidades de modelado. Ejemplos de las mismas pueden ser las metaclases *Class*, *Association*, etc.

Por otro lado, la relación existente entre dos niveles consecutivos es una relación de instanciación. La misma puede darse en el modelo (representada en el diagrama con flechas de línea punteada) o en el metamodelo (representada con flechas de línea completa). Por ejemplo, *Robert* es instancia de la clase *Client* (instanciación en el modelo), que, a su vez, es instancia de la metaclase *Class* (instanciación en el metamodelo).

Dado que lo que se quiere plantear en este capítulo es una representación clara de las etapas de CHDM, fue necesario elaborar un *modelo* para cada una de ellas. Ahora bien, estos modelos también pueden ser considerados metamodelos de aplicaciones colaborativas puesto que capturan la esencia de todas las aplicaciones colaborativas, son modelo de modelos.

En este trabajo de grado es preciso aclarar que se ha trabajado en los niveles de modelo y de metamodelo. En este último para plantear las distintas etapas de CHDM, y en el nivel de modelo para ejemplificar e instanciar cada etapa en un dominio concreto, es decir, en un modelo concreto.

3.3.1 Modelo Conceptual

Como se dijo anteriormente, en esta etapa se identifican los objetos relevantes de la aplicación y las relaciones entre ellos. El modelo conceptual es como cualquier diseño de objetos con la salvedad de que los atributos pueden ser multitipados, o sea, que pueden pertenecer a más de un tipo o clase. Por ejemplo, en un diseño preliminar para una aplicación de un museo de arte, la representación de un pintor podría tener un atributo que modelara su biografía. La misma podría representarse por medio de texto o bien de video. Pero esta decisión podría delegarse a la etapa navegacional. Es por eso que en la etapa conceptual el atributo biografía es multitipado.

En la figura 3.2 se muestra el modelo de la etapa de diseño conceptual de CHDM a través de un diagrama de clases UML y a continuación se da una breve descripción de cada una de las metaclases. Cabe aclarar que esta etapa es exactamente igual a la que plantea OOHDM. Un aporte de este trabajo de grado fue plasmar los conceptos de esta etapa descripta en [Ros96] en un diagrama de metaclases. A su vez se han tomado ciertas libertades a la hora de especificar los tipos de relaciones entre clases (ver jerarquías de *Relationship* y *KnowledgeStrategy*). Todas las metaclases, excepto *Class*, *Relationship* y *Attribute* fueron agregadas para una mejor conceptualización

de las entidades intervinientes en esta etapa de diseño.

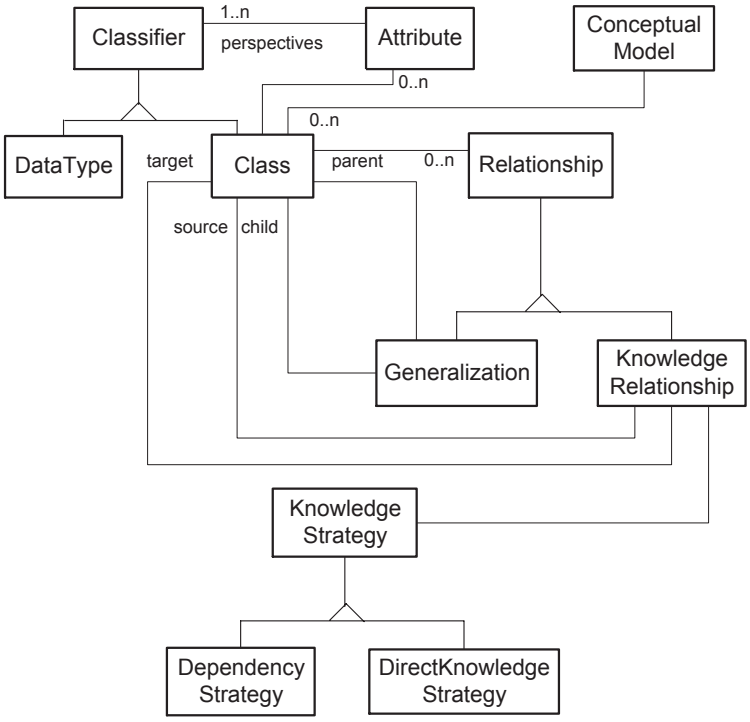

Figura 3.2: Modelo Conceptual de CHDM

La metaclase *ConceptualModel* representa a los modelos conceptuales de los diseños de aplicaciones de hipermedia. La misma está relacionada con la clase *Class* puesto que un modelo conceptual está compuesto por un conjunto de clases. La jerarquía de *Classifier* modela a los objetos de una aplicación que se pueden representar mediante clases *Class* o tipos de datos *DataType*. La metaclase *Attribute* representa a los atributos de los *Classifiers* y tiene una relación a la metaclase *Classifier* con multiplicidad $1..n$, pues, como se dijo anteriormente, los atributos pueden ser multitipados en esta etapa.

Por otro lado, se tiene la jerarquía de *Relationship* que representa a las posibles relaciones entre objetos. Tales relaciones pueden ser de generalización, *GeneralizationRelationship*, o de conocimiento *KnowledgeRelationship*. Las relaciones de conocimiento, a su vez, pueden ser de diferente naturaleza: relaciones de conocimiento simples, dependencias, etc; es por eso que llevan asociadas una estrategia de conocimiento *KnowledgeStrategy*, haciendo uso del patrón de diseño *Strategy* [GHJV94, p. 315], que determina las características de tales relaciones.

3.3.2 Modelo Navegacional

En la etapa del modelo navegacional se definen los nodos, links, anchors, contextos navegacionales, que van a conformar el espacio navegacional de la aplicación.

Aquí, al igual que en la etapa de diseño conceptual, cabe aclarar que CHDM reusa la etapa de diseño navegacional de OOHDM. De manera análoga al diseño conceptual se ha elaborado un diagrama de clases UML para representar los conceptos definidos en [Ros96] en un único diagrama que los interrelacione. Se han incluido detalles de diseño tales como *LinkEndPoint* y la jerarquía de *EndPointSolver* definidas en [RG96], que tienen como objetivo desacoplar los links de los destinos de links.

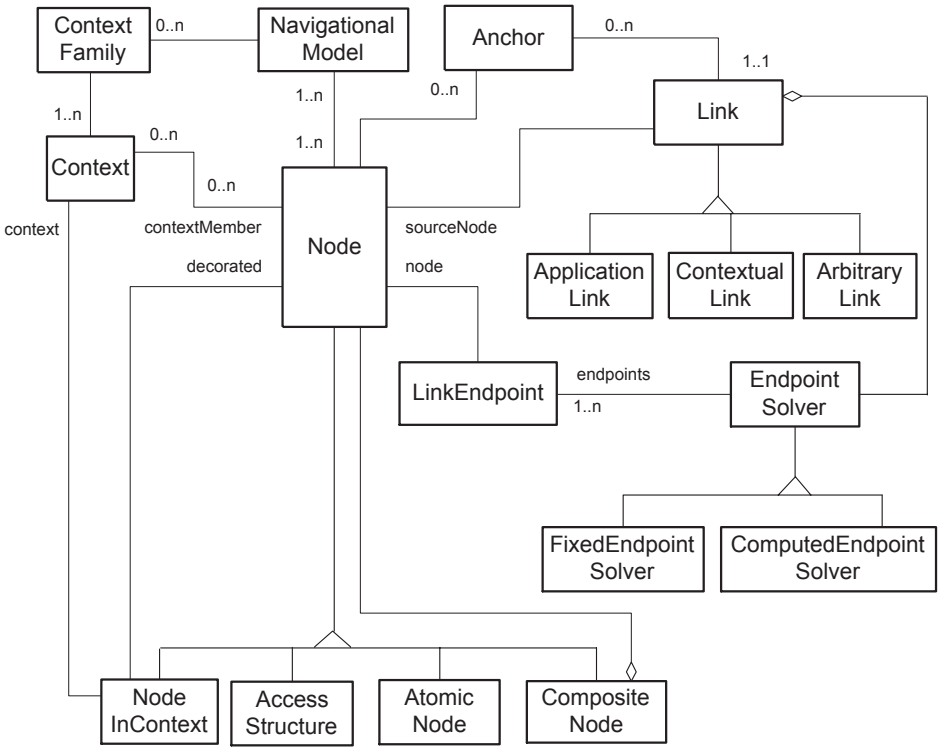

Figura 3.3: Modelo Navegacional de CHDM

En la figura 3.3 se muestra el modelo navegacional de CHDM. En la misma se muestra la metaclase *Node* que representa a las clases de nodos definidas para cualquier modelo navegacional. Existen cuatro clases de nodos: *AtomicNode* que representa a los nodos atómicos, *CompositeNode* que representa a los nodos compuestos, *AccessStructure* que representa a las es-

tructuras de acceso, tales como índices a otros nodos, y *NodeInContext* que es un *decorador* [GHJV94, p.175] de los nodos para agregar información del contexto de navegación en que se encuentra el nodo *decorado*. Cualquier nodo, ya sea atómico, compuesto, estructura de acceso o *InContext*, conoce un conjunto de *Anchors*, que representan los puntos de partida para navegar a través de los links a otros nodos.

La jerarquía de *Link* representa las uniones entre los nodos. A través de ellos se puede navegar de un nodo a otro. Se han definido tres subclases: *ApplicationLink*, que representa a los links que surgen de las relaciones de conocimiento, *ContextualLink*, que representa a las relaciones entre nodos que surgen implícitamente a partir de la definición de contextos navegacionales, y por último, *ArbitraryLink* para modelar relaciones arbitrarias entre nodos.

A su vez, los links están relacionados con un *EndPointSolver* que es el encargado de decidir el destino al que se navegará mediante ese link. *LinkEndPoint* fue considerada para permitir destinos mútliples a partir de un mismo link. En este caso, el *EndPointSolver* brindaría un conjunto de *LinkEndPoints* de los cuales el usuario seleccionaría el próximo nodo a navegar. El destino puede ser fijo o computado en tiempo de ejecución, para lo cual se tienen las subclases *FixedEndPointSolver* y *ComputedEndPointSolver*.

3.4 Modelo Colaborativo

Como fue mencionado anteriormente, la etapa del modelo colaborativo fue propuesta con el fin de agregar a OOHDM características para soporte de diseño de aplicaciones de hipermedia colaborativas.

Como resultado de la nueva etapa, se obtiene un modelo de objetos concreto para especificar de manera estructural, o estática, las características colaborativas de la aplicación, y diagramas de secuencia que permiten especificar la parte dinámica de la aplicación, es decir, el comportamiento de dichos objetos en escenarios colaborativos puntuales y la manera en que ellos colaboran.

En lo que respecta al usuario de CHDM, un diseñador de aplicaciones de hipermedia colaborativas, a la hora de construir el modelo de una aplicación de hipermedia colaborativa solamente necesita instanciar cada uno de los modelos de CHDM con las clases específicas de su aplicación.

En la sección 3.4.1 se mostrará el diseño del modelo colaborativo enfocado desde el punto de vista estructural y en la sección 3.4.2 se describen los aspectos dinámicos del mismo. Finalmente en la sección 3.4.3 se mostrará

un ejemplo de la instanciación del modelo colaborativo para una aplicación concreta: un laboratorio de investigación.

3.4.1 Aspectos Estructurales

En esta sección se describirán los aspectos estructurales del modelo colaborativo. Es decir, se mostrarán las clases participantes y sus relaciones, pero no el modo en que ellas interactúan.

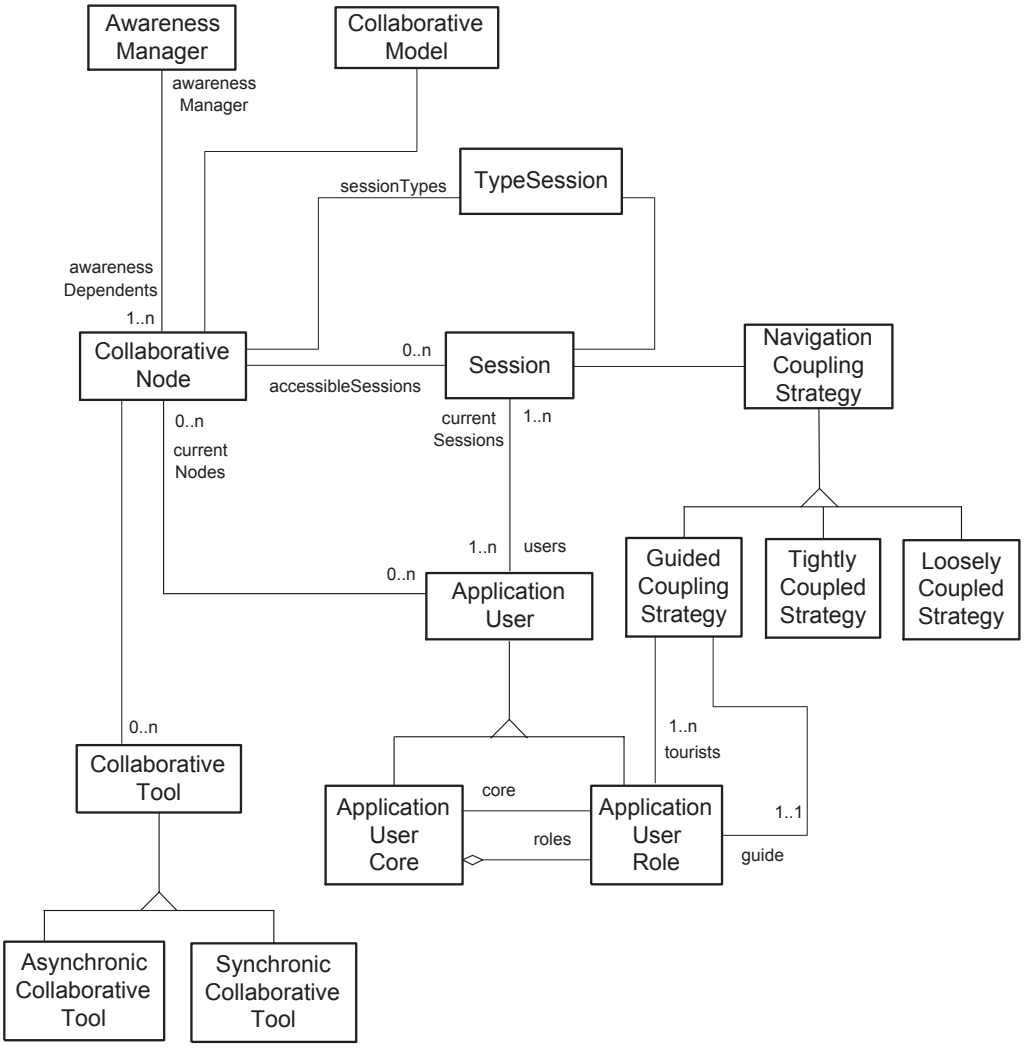

Figura 3.4: Modelo Colaborativo de CHDM

La figura 3.4 muestra las metaclases que intervienen en el modelo cola-

borativo.

Uno de los aspectos a tener en cuenta para el diseño de aplicaciones de hipermedia colaborativas es el de usuarios y sus roles en la utilización de la aplicación. Esto se encuentra representado en el modelo colaborativo por la jerarquía de *ApplicationUser*. Aquí, nuevamente se ha utilizado un patrón de diseño orientado a objetos, en este caso el patrón *Role Object* [BRSW00], dado que permite agregar roles dinámicamente a objetos. Los usuarios están relacionados con las sesiones de colaboración en las que se encuentran trabajando.

La metaclase *CollaborativeModel* representa a los modelos colaborativos de CHDM. Ésta conoce a un conjunto de *CollaborativeNode*, nodos colaborativos que componen el modelo. En la sección 3.5 se verá que el nodo colaborativo es el punto de conexión entre el modelo colaborativo y el modelo navegacional, puesto que cada nodo colaborativo *decora* a un nodo del modelo navegacional, agregándole cualidades colaborativas (patrón *Decorator* [GHJV94, p.175]).

Una de las características colaborativas, el *awareness*, está modelada a través del *AwarenessManager*. Éste distribuye la información de *awareness* entre nodos colaborativos que la requieran. Los nodos colaborativos se deben "suscribir" como dependientes del *AwarenessManager* para recibir notificaciones de los eventos en los que estén interesados y deben notificarle sobre los eventos relevantes. Así, por ejemplo, un nodo que quiera proveer *awareness* de presencia de los usuarios estará interesado en saber cuándo un usuario entra o sale de la aplicación; dicha información será provista al *AwarenessManager* por parte del nodo en el que el usuario se loguea o desloguea de la aplicación.

Por su parte, las metaclases *TypeSession* y *Session* representan los tipos de sesiones que un nodo colaborativo ofrece a los usuarios y las sesiones de colaboración concretas (ya iniciadas), respectivamente (metaclases inspiradas en el patrón de diseño *Type Object* [JW97]). Por ejemplo, un nodo puede brindar la posibilidad de entablar una sesión de discusión entre usuarios (tipo de sesión) y a su vez, permitir a un usuario ingresar a una discusión donde un grupo de usuarios ya se encuentra colaborando (sesión concreta).

Por otro lado, la metaclase *NavigationCouplingStrategy* representa la estrategia de acoplamiento utilizada en las sesiones de navegación (pattern Strategy, [GHJV94, p.315]). En la figura 3.4 la jerarquía encabezada por *NavigationCouplingStrategy* tiene tres subclases: *GuidedCouplingStrategy*, que apunta al diseño de navegaciones guiadas, *TightlyCoupledStrategy*, en donde todos los usuarios navegan juntos, y *LooselyCoupledStrategy*, en donde los usuarios navegan independientemente de los demás. La metodología CHDM propone que esta jerarquía sea ampliada de acuerdo a los requeri-

mientos de la aplicación a ser diseñada.

Además, el modelo colaborativo de CHDM brinda la posibilidad de agregar herramientas colaborativas a los nodos. Éstas pueden ser sincrónicas, tal como una herramienta de *chat*, o asincrónicas, como la posibilidad de enviar mensajes entre usuarios o compartir información a través de archivos. Éstas se ven representadas mediante la jerarquía *CollaborativeTool*. Los detalles de diseño de tales herramientas están fuera del ámbito de este trabajo, puesto que cada una de ellas es una aplicación en sí misma.

3.4.2 Aspectos Dinámicos

En esta sección se mostrará el modelo colaborativo en sus aspectos dinámicos. Es decir, se verá el modo en que interactúan las diferentes clases que lo componen. Para tal fin, se mostrarán diferentes diagramas de secuencia para algunos escenarios de colaboración entre objetos. Notar que los siguientes diagramas son *metadiagramas* dado que especifican interacciones a nivel de metaclases, o sea, abstraen interacciones entre clases de cualquier modelo.

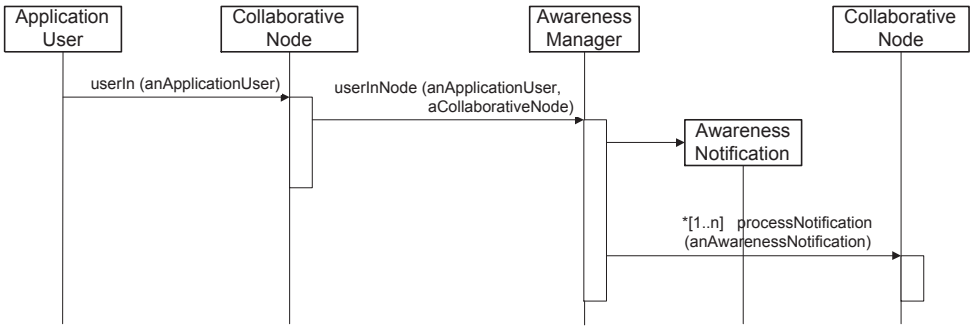

Figura 3.5: Llegada a un nodo colaborativo

En la figura 3.5 se muestra la secuencia de acciones que se llevan a cabo cuando un usuario llega a un nodo colaborativo. Las dos formas en que un usuario puede llegar a un nodo colaborativo son: habiéndose logueado en ese nodo (primer nodo de la aplicación que visita), o habiendo navegado desde otro nodo.

Cuando un usuario llega a un nodo colaborativo se genera un evento que se expande a todos los demás nodos colaborativos de la aplicación mediante el *AwarenessManager* que es el encargado de notificar a los nodos los eventos de *awareness* para que actualicen la información correspondiente.

En la figura 3.6 se muestra la secuencia de un usuario cuando entra en una sesión de colaboración. Una vez que el usuario elige una sesión a la

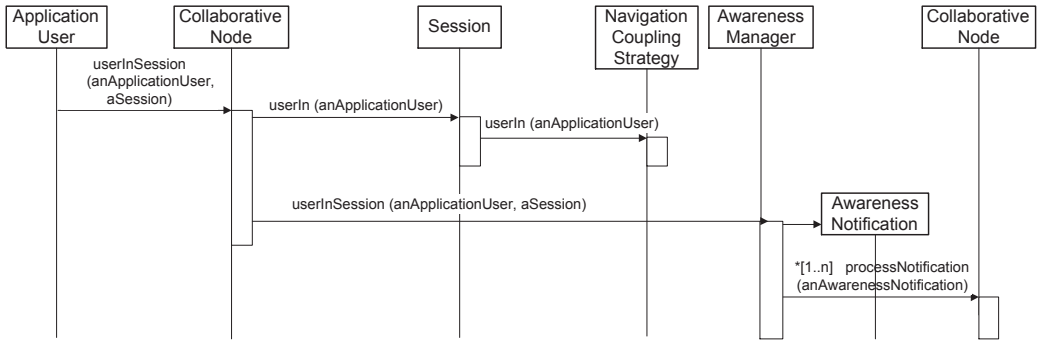

Figura 3.6: Entrada a una sesión de colaboración

cual ingresar, se lo comunica al nodo colaborativo por medio del cual ingresará. Éste se comunica con la sesión y ella, a su vez, con la estrategia de navegación asociada para que tenga en cuenta al nuevo usuario para las navegaciones de la sesión. De la misma manera que en el diagrama anterior, el *AwarenessManager* se encarga de distribuir el evento a todos los nodos colaborativos de la aplicación.

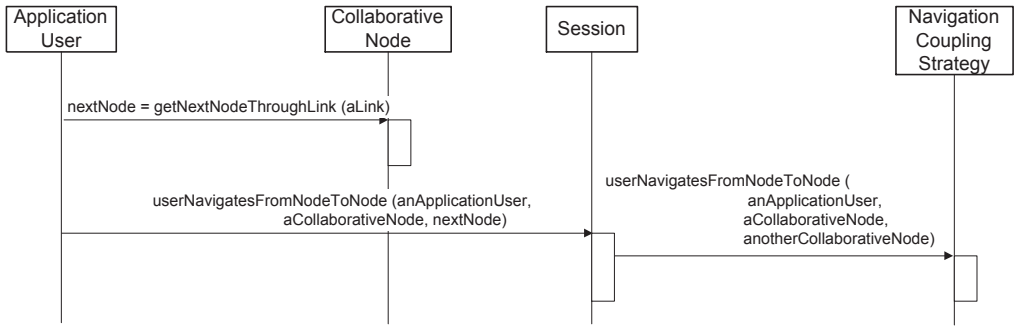

Figura 3.7: Navegación

La figura 3.7 muestra el escenario de un usuario que navega de un nodo a otro a través de un link. El usuario le pide al nodo el siguiente nodo correspondiente al link y luego le comunica a la sesión que quiere navegar. Ésta determina los pasos a seguir en colaboración con la estrategia de navegación *NavigationCouplingStrategy* ya que la navegación de un usuario puede afectar al resto de los usuarios de la sesión.

En la figura 3.8 se muestra lo que sucede cuando un usuario navega en una sesión fuertemente acoplada: los demás usuarios navegan con él. La sesión le avisa a su estrategia de navegación que un usuario quiere navegar. La

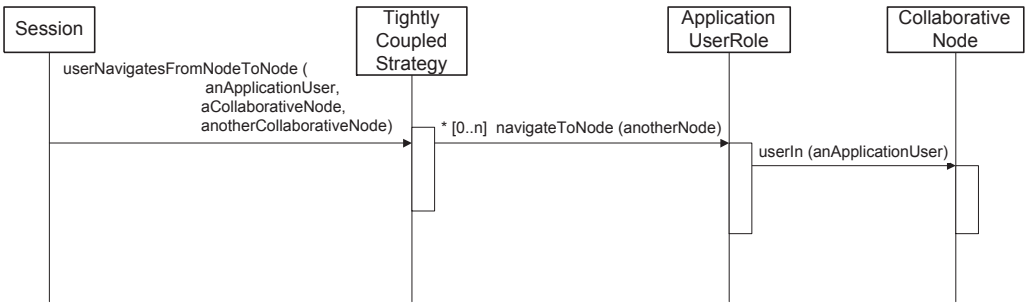

Figura 3.8: Navegación en una sesión fuertemente acoplada

estrategia de navegación, al ser fuertemente acoplada, se encarga de comunicar a todos los usuarios de la sesión (incluyendo al que originó la navegación) que tienen que navegar hacia otro nodo.

3.4.3 Ejemplo

En esta sección se dará un ejemplo de la utilización de la etapa del metamodelo colaborativo propuesto en una aplicación concreta. Se mostrará cómo, para construir un diseño de la etapa colaborativa de CHDM sólo es necesario instanciar el metamodelo colaborativo propuesto.

La figura 3.9 representa el diseño de la etapa colaborativa de una aplicación de un laboratorio de investigación. En tal aplicación, es necesario representar entidades tales como áreas de investigación, investigadores, proyectos de investigación que se desarrollan, etc. En el diseño conceptual intervienen, entre otras, las clases *ResearchArea*, *LabMember* y *Project*, propias del dominio de la aplicación. Dichas clases son instancias de la metaclase *Class* del metamodelo conceptual de CHDM. Suponiendo que en el diseño navegacional se tiene, entre otras, una instancia de la metaclase *Node* (del metamodelo navegacional) que representa el mapa de la aplicación, *SiteMapNode*, lo que se pretende modelar en este diseño colaborativo es la posibilidad de realizar una navegación acoplada en la que un usuario que es miembro del laboratorio sea el guía de dicha navegación, y el resto de los usuarios lo sigan a manera de tour.

Como se mencionó anteriormente, para definir el modelo colaborativo de una aplicación en particular sólo es necesario instanciar el metamodelo colaborativo de CHDM con las clases del dominio de la aplicación. En este caso, se tiene un nodo colaborativo, *CollaborativeSiteMapNode*, que refleja el nodo correspondiente del nivel navegacional pero además provee la posibilidad de que los usuarios se agreguen a una sesión de navegación guiada.

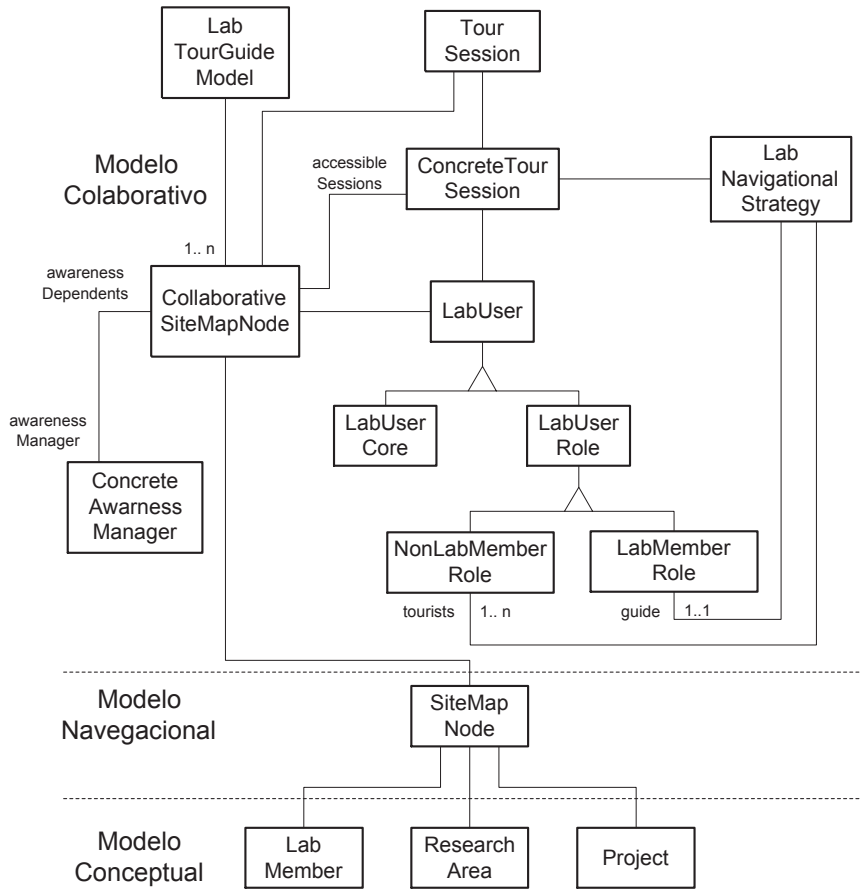

Figura 3.9: Ejemplo de Modelo Colaborativo

Es decir, si el guía navega de un nodo a otro, todos los usuarios que participan en esa misma sesión navegarán al nodo que el guía indique. La idea es que el guía sea un miembro del laboratorio que muestra a otros usuarios (externos al laboratorio) un tour por la aplicación. El nodo colaborativo *CollaborativeSiteMapNode* es instancia de la metaclase *CollaborativeNode* del metamodelo colaborativo propuesto por CHDM.

Además se ha especificado que el nodo colaborativo definido utilice un *AwarenessManager* concreto denominado *ConcreteAwarenessManager* (instancia de la metaclase *AwarenessManager* del metamodelo de CHDM) que se encargue de los aspectos de implementación del awareness del nodo. No se dan detalles de diseño sobre el objeto *ConcreteAwarenessManager*, pues para tal fin es necesario mostrar un diseño más completo, y aquí sólo se quiere mostrar un ejemplo.

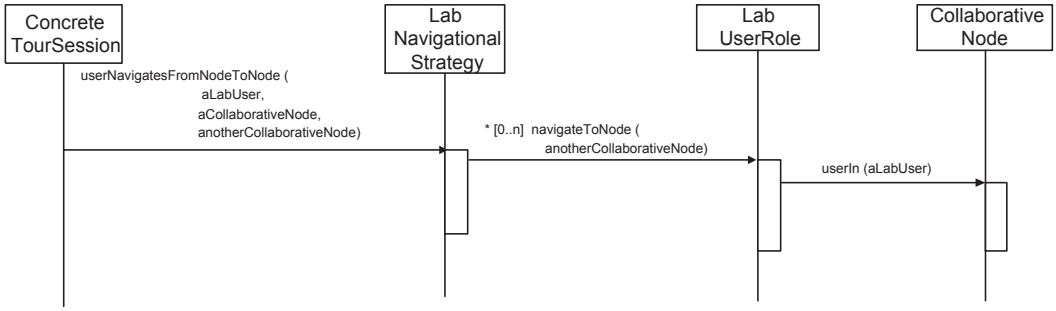

Figura 3.10: Instanciación de un Diagrama de Secuencia

La figura 3.10 muestra el diagrama de secuencia que surge al haber instanciado el modelo colaborativo de CHDM para una aplicación concreta. En este caso se puede ver la secuencia de acciones relacionadas con la navegación acoplada mencionada anteriormente. Este es un ejemplo de modelización de aspectos dinámicos de una aplicación de hipermedia colaborativa.

Es notable ver cómo el metamodelo colaborativo planteado anteriormente (figura 3.4) sirve como "molde" para instanciar un modelo concreto (figura 3.9). Basta superponer ambos diagramas para ver las instanciaciones de las clases intervinientes. Por ejemplo, *CollaborativeSiteMapNode* es instancia de *CollaborativeNode*, *LabUser* lo es de *ApplicationUser* y así sucesivamente. La estrategia de navegación *LabNavigationalStrategy* fue instanciada, para este ejemplo, a partir de la metaclase *GuidedCouplingStrategy* puesto que la navegación acoplada cumple la metáfora de "navegación guiada". Para representar los distintos roles en la sesión de navegación se utilizan las clases *LabMemberRole* y *NonLabMemberRole*.

En conclusión, para desarrollar un modelo colaborativo concreto, es necesario instanciar las metaclases propuestas en la metodología CHDM con las clases propias del dominio de aplicación.

3.5 Relación entre los Modelos

En esta sección se verá cuáles son las relaciones entre los modelos conceptual, navegacional y colaborativo. Luego, en la sección 3.5.1 se mostrará un ejemplo concreto instanciando cada uno de los modelos de CHDM para una aplicación de un museo de arte.

Para comenzar, en la figura 3.11, se muestran las conexiones existentes entre los modelos conceptual, navegacional y colaborativo. Existen dos relaciones entre los modelos conceptual y navegacional; la primera es entre las me-

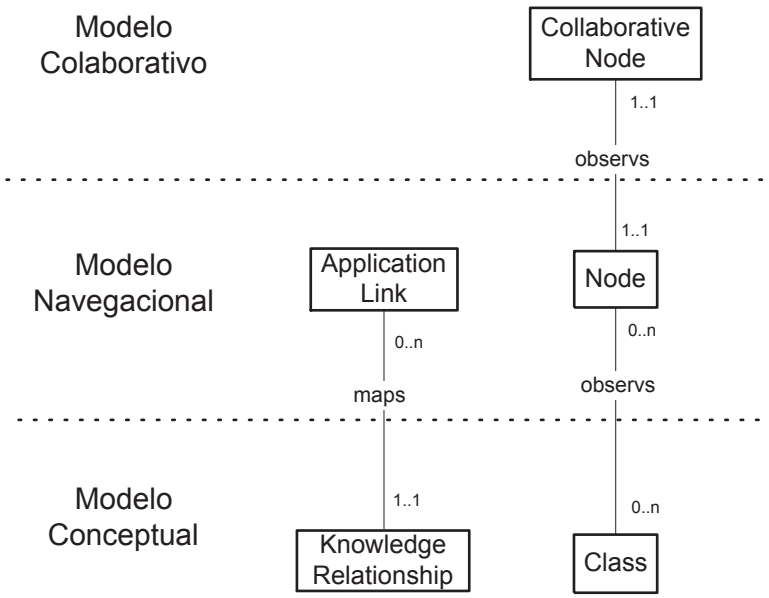

Figura 3.11: Relación entre los Modelos

taclases *Class* y *Node* de sendos modelos. Entre ellas se presenta una relación de dependencia: un nodo observa a varios objetos de los cuales es dependiente, y de los cuales obtiene atributos para mostrar. La segunda conexión entre ambos modelos se encuentra entre las metaclases *KnowledgeRelationship* y *ApplicationLink*. En este caso, un link de la aplicación mapea una relación existente entre objetos de la aplicación. Por ejemplo, la relación *pinta* entre las clases *Pintor* y *Pintura* en un modelo conceptual se vería reflejada en un link que permitiera navegar de un pintor a una de sus pinturas, en un modelo navegacional.

Finalmente, se muestra la relación existente entre el modelo navegacional y el colaborativo reflejada mediante las metaclases *CollaborativeNode* y *Node*. La relación entre ambos cumple con el pattern *Decorator* [GHJV94, p.175] puesto que el nodo colaborativo agrega al nodo ciertas características colaborativas, tales como *awareness*, acoplamiento en la navegación, grupos de usuarios colaborando, herramientas colaborativas, etc.

Notar que la metaclase *ApplicationLink* del modelo navegacional no tiene una representación en el modelo colaborativo como la metaclase *Node*, pues los links no requieren características colaborativas como los nodos. Éstos son accesibles mediante los nodos colaborativos que realizan la conexión con el modelo navegacional. Cuando un usuario selecciona un anchor en un nodo colaborativo, en realidad está seleccionando la "vista" que tiene el nodo cola-

borativo de tal anchor, y el resultado de tal acción dependerá de la estrategia de navegación asociada al nodo colaborativo, conjuntamente con el rol que esté desempeñando el usuario en el sistema. Por ejemplo, si el usuario se encuentra en una sesión de tour por la aplicación y tiene el rol de guía, todos los demás usuarios de la misma sesión serán forzados a navegar con él. Pero para esta nueva metáfora de navegación no es necesario modificar ni agregar ninguna característica en especial a los anchors ni a los links, ellos funcionan de la misma manera que en una aplicación no colaborativa, el destino del link es el mismo.

3.5.1 Ejemplo

En esta sección, se ejemplificarán las relaciones existentes entre los modelos conceptual, navegacional y colaborativo mostrando un modelo de una aplicación de hipermedia colaborativa concreta: un museo de arte. En tal aplicación encontraremos objetos tales como pintores y pinturas, para los cuales es necesario modelar nodos. Además se agrega la funcionalidad colaborativa de poder discutir con otros usuarios visitantes temas relacionados con un pintor al llegar al nodo correspondiente al artista. Para eso los visitantes deberán entablar una sesión de discusión.

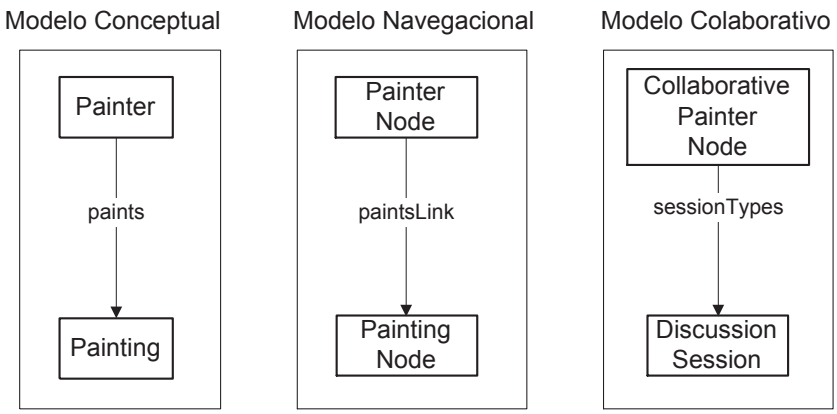

Figura 3.12: Modelo del Museo de Arte

En la figura 3.12 se brinda un pequeño ejemplo de los modelos conceptual, navegacional y colaborativo de CHDM instanciados para la aplicación del museo de arte anteriormente mencionada. En el modelo conceptual se tienen las clases *Painter* y *Painting* unidas por la relación *paints* que representa la relación entre un pintor y sus pinturas. En el modelo navegacional, por

otro lado, se han modelado los nodos correspondientes a los pintores y las pinturas mediante *PainterNode* y *PaintingNode* entre los cuales se encuentra el link *paintsLink* representando la posibilidad de navegar de un pintor a una de sus pinturas. Por último, en el modelo colaborativo se tienen las clases *CollaborativePainterNode* y *DiscussionSession* que representan que el nodo colaborativo agrega al nodo de un pintor la posibilidad de establecer una sesión de discusión con otros usuarios.

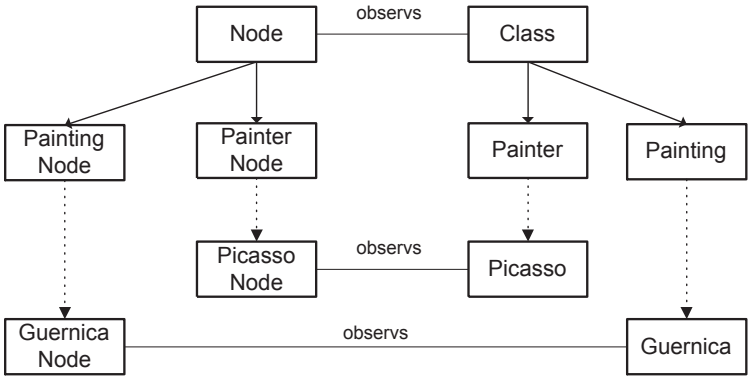

Figura 3.13: Niveles de Instanciación para Nodos

En la figura 3.13 se muestran los diferentes niveles de instanciación como se describió en la sección 3.3 y a su vez, la relación existente entre el modelo conceptual y navegacional de CHDM.

En la parte superior del diagrama se observan las metaclases *Class* y *Node* (metaclases del modelo de CHDM). A partir de ellas se instancian *PaintingNode*, *PainterNode*, *Painter* y *Painting* que son clases del modelo de la aplicación (instanciación en el metamodelo, flecha de línea completa). Y finalmente, se obtienen los objetos de la aplicación *Picasso*, *Guernica*, *PicassoNode* y *GuernicaNode* como instancias de tales clases (instanciación en el modelo, flecha de línea punteada).

Nótese además, que las relaciones *observs* que se daban entre las metaclases *Node* y *Class* en el metamodelo, son transportadas hacia las instancias del modelo dado que el metamodelo es una abstracción del modelo, que a su vez, es una abstracción del dominio de la aplicación.

En la figura 3.14 se muestran nuevamente los diferentes niveles de instanciación, pero en este caso con respecto a los modelos navegacional y colaborativo de CHDM, y a la relación entre ellos. Las metaclases *Node* y *CollaborativeNode* son instanciadas para obtener las clases del modelo *PainterNode* y *PainterCollaborativeNode* de las cuales se obtienen las instan-

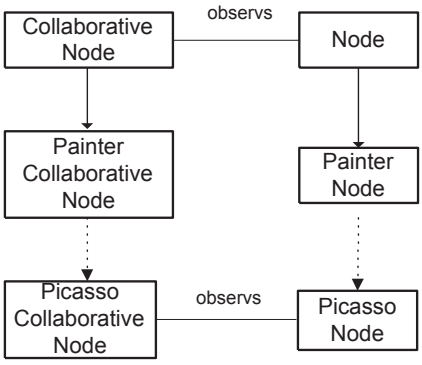

Figura 3.14: Niveles de Instanciación

cias *PicassoNode* y *PicassoCollaborativeNode* que son objetos de la aplicación del museo de arte.

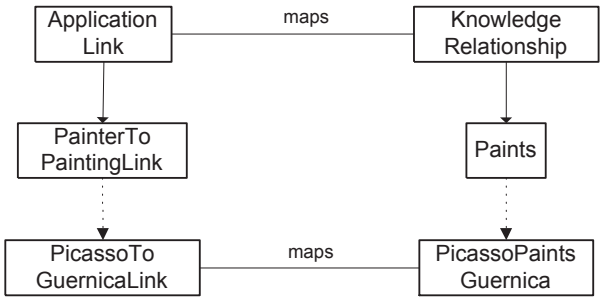

Figura 3.15: Niveles de Instanciación para Links

En la figura 3.15 se pueden observar los tres niveles de abstracción y las relaciones de instanciación entre ellos, en el eje vertical, y una de las relaciones entre los modelos conceptual y navegacional (entre *KnowledgeRelationship* y *Link*), en el eje horizontal. Puede verse que la relación especificada a nivel metamodelo *maps*, se traslada hacia el modelo representando que un link mapea la relación entre un pintor y su pintura.

3.6 Contribuciones de CHDM

En la sección 3.2.1 se mostraron las características de las cuales OOHDM carecía para soportar el diseño de aplicaciones de hipermedia colaborativas.

A continuación se describe en qué manera CHDM soluciona cada uno de los puntos discutidos en dicha sección:

1. CHDM define una jerarquía de usuarios y roles de usuarios que debe ser instanciada de acuerdo a las necesidades de la aplicación. Además permite especificar características colaborativas tales como *awareness* para cada nodo colaborativo, asociándole un manejador de *awareness*, instancia de la metaclase *AwarenessManager*.

2. CHDM permite asociar a los nodos colaborativos herramientas colaborativas sincrónicas o asincrónicas.

3. CHDM permite modelar sesiones de interacción y colaboración entre usuarios. Para eso propone la definición de tipos de sesiones accesibles a partir de los diferentes nodos colaborativos de la aplicación.

El diseñador de aplicaciones de hipermedia colaborativas sólo tiene que instanciar el metamodelo propuesto por CHDM para modelar las características colaborativas mencionadas en los puntos anteriores.

En conclusión, CHDM extiende la metodología de diseño OOHDM para aplicaciones de hipermedia colaborativas, modelando las características colaborativas de tales aplicaciones que OOHDM no tiene en cuenta.

Capítulo 4

Formalización

Una técnica bastante aceptada en ingeniería de software es la combinación de diferentes modelos para describir distintos aspectos (de estructura y comportamiento) de un dominio de aplicación. El hecho de usar diferentes modelos aclara partes importantes del sistema, pero debe tenerse en cuenta que los mismos pueden no ser independientes y entonces pueden superponerse semánticamente. Los modelos constituyen la base fundamental de información sobre la cual interactúan expertos del dominio, analistas y desarrolladores de software. Por lo tanto, es de suma importancia que los modelos expresen claramente la esencia del problema [PB00].

Por otro lado, la actividad de construcción de un modelo es una parte crítica en el proceso de desarrollo de software [SM88]. Dado que los modelos son el resultado de un proceso complejo de abstracción y creación, es posible que tiendan a contener errores, omisiones e inconsistencias. Los mismos se construyen usando un lenguaje de especificación, que puede variar desde lenguaje natural (no formal) a lenguajes de especificación gráficos, como UML [Gro99] (semi-formal) y a veces hasta un formalismo matemático (formal).

El éxito de los lenguajes de especificación gráficos, como UML o Booch [Boo94] se basa en el uso de elementos de modelado intuitivos y mecanismos estructurados, que son fáciles de entender, aplicar y transmitir. Sin embargo, la falta de semántica precisa de las notaciones de modelado utilizadas por este tipo de lenguajes puede ocasionar varios problemas:

- Malas interpretaciones de los modelos: la interpretación del lector podría ser diferente a la interpretación del creador del modelo.
- Existencia de varios modelos separados (estáticos, dinámicos, funcionales) difíciles de integrar y mantener consistentes.
- Los modelos no pueden validarse (ni formal ni informalmente), lo cual

puede llevar a sistemas inseguros donde, por ejemplo, el comportamiento bajo ciertas circunstancias sea impredecible.

En [CMPR00] se describen en detalle los problemas de ambigüedades contenidos en UML y se propone una base formal al mismo. En particular, se tratan los diagramas de colaboración en sus aspectos estáticos (sintácticos) y dinámicos (semánticos).

Por otro lado, los lenguajes de especificación formales, tales como Z [DKRS91, Spi92], VDM [Jon90] o F-Logic [MK90], tienen una sintaxis y semántica bien definidas. Si bien su uso en la industria es poco frecuente debido a la complejidad de los formalismos matemáticos que utilizan, tienen mayor poder expresivo y están faltos de ambigüedades.

En el capítulo 3 se mostró una formalización mediante diagramas de objetos en notación UML para la metodología de diseño CHDM. Al ser UML un lenguaje de especificación gráfico semi-formal cuenta con las desventajas anteriormente descriptas de tales lenguajes, por lo tanto en este capítulo se dará una especificación de CHDM mediante un lenguaje formal (Object-Z), permitiendo así que los modelos de aplicaciones de hipermedia colaborativas diseñados con la metodología CHDM estén más precisamente especificados.

Object-Z cuenta con varias ventajas con respecto a UML por ser un lenguaje formal. A continuación se describen algunas de ellas:

- Object-Z es un lenguaje de especificación más expresivo y no ambiguo.
- Al ser orientado a objetos, Object-Z cuenta con todas las ventajas de ese paradigma.
- Object-Z permite especificar reglas de buena formación para los esquemas de clase, invariantes que deben cumplir todas las instancias de tal clase.
- Object-Z permite especificar las operaciones en forma declarativa.
- Se pueden realizar chequeos sobre las clases especificadas, de acuerdo a los invariantes y operaciones definidas.
- Permite derivar nuevas propiedades a partir de las ya especificadas y de las reglas de buena formación correspondientes a cada clase.

Como corolario, por haber especificado formalmente el modelo de CHDM con Object-Z, surgen las siguientes ventajas:

- La especificación en Object-Z provee una base formal a la metodología CHDM.

- Como CHDM conforma un metamodelo para las aplicaciones de hipermedia colaborativas, dicho metamodelo será instanciado a la hora de realizar un modelo para una aplicación en particular. Una vez instanciado el modelo de CHDM puede verificarse el cumplimiento de las reglas de buena formación especificadas para las metaclases. Por lo tanto, se provee una base formal para todos los diseños de aplicaciones de hipermedia colaborativas.

- Podría implementarse un editor de CHDM que permitiera verificaciones dentro de los modelos generados por éste basándose en el formalismo subyacente y los invariantes especificados.

En las secciones siguientes se mostrarán algunas metaclases elegidas, las consideradas más representativas de cada etapa, con el propósito de ejemplificar la especificación formal del modelo CHDM. En la sección 4.1 se verán algunas de las metaclases del modelo navegacional, y en la sección 4.2 metaclases del modelo colaborativo. Finalmente en la sección 4.3 se mostrará un ejemplo donde se verá la instanciación de las metaclases formalizadas, para una aplicación de hipermedia colaborativa concreta.

No hemos elegido ninguna clase del modelo conceptual por tratarse de un modelo de objetos con la única particularidad de que los atributos de las clases pueden ser multi-tipados. Tampoco hemos considerado a las etapas de interfase abstracta ni la de implementación, puesto que tampoco fueron consideradas en la definición del modelo de CHDM en el capítulo 3. Estas últimas dos etapas podrían tratarse en una futura investigación sobre el tema.

Se recomienda al lector no familiarizado con la notación de Object-Z remitirse al apéndice B antes de continuar con el resto de este capítulo.

4.1 Modelo Navegacional

En esta sección se mostrarán algunas de las metaclases del modelo navegacional de CHDM especificadas con el lenguaje formal Object-Z. Tal modelo fue presentado en la sección 2.2 como se describe en la definición original de OOHDM, y más tarde, en la sección 3.3.2 se mostró un diagrama de objetos correspondiente a dicho modelo. Se verán las metaclases: *NavigationalModel*, *Link*, *Node*, *NodeInContext*, *Context* y *ContextFamily* que han sido seleccionadas por considerarse las más significativas del modelo navegacional.

NavigationalModel

$nodes : \mathbb{P}\, Node$
$contextFamilies : \mathbb{P}\, ContextFamily$
$anchors : \mathbb{P}\, Anchor$

$anchors = \{a \in Anchor \mid \exists\, n \in nodes \bullet a \in n.anchors\}$

---

getNavigationalPaths

$paths! : \mathbb{P} \operatorname{seq} Node$

$\forall\, p \in paths! \bullet ((\forall\, i \in \operatorname{dom} p \bullet p[i] \in nodes)$
$\qquad \wedge\, (p[i].isNavigableTo(p[i+1])))$

---

existsPath

$sourceNode? : Node$
$targetNode? : Node$

$self.getNavigationalPaths \;{}^{\circ}_{9}$
$\exists\, p \in paths! \bullet$
$\qquad paths![1] = sourceNode? \wedge$
$\qquad paths![paths!.length] = targetNode?$

$[1]\, \forall\, a \in anchors \bullet a.getDestination() \in nodes$

Figura 4.1: NavigationalModel

En la figura 4.1 se describe la especificación en Object-Z de la metaclase *NavigationalModel*. Ésta representa a los modelos navegacionales de OOHDM y abstrae las características y comportamiento comunes a los mismos. Sus atributos son: *nodes*, los nodos pertenecientes al modelo navegacional, *contextFamilies*, las familias de contextos que conforman la estructura

navegacional de dicho modelo y *anchors*, un atributo derivado que representa a todos los anchors de los nodos del modelo. Se han especificado las operaciones *getNavigationalPaths*, que retorna todos los posibles caminos de navegación entre los nodos de la aplicación, y *existsPath*, que dice si existe un camino navegable entre dos nodos dados. Para esto hace uso de la operación anterior utilizando su variable resultado *paths*!. Además se definió la regla de buena formación [1] que especifica que todos los anchors de los nodos del modelo navegacional conectan nodos pertenecientes a dicho modelo. Recordar que en Object-Z los parámetros de entrada se marcan con el símbolo "?" y los de salida con "!" (ver apéndice B).

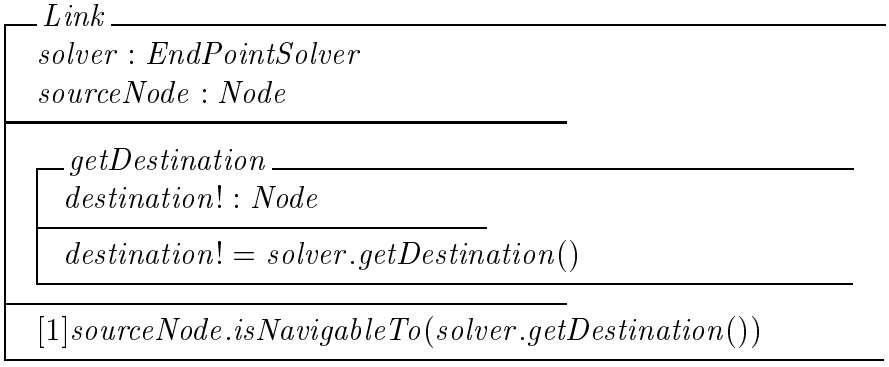

Figura 4.2: Link

En la figura 4.2 se muestra la especificación formal de la metaclase *Link*. La misma representa a las conexiones que permiten la navegación de un nodo a otro. Cuenta con los atributos *solver* y *sourceNode* que representan al objeto encargado de calcular el destino del link y el nodo origen del link, repectivamente. La operación *getDestination* se encarga de retornar el nodo destino del link, colaborando con el *solver*. Además, se especifica el invariante [1] que establece que a partir del nodo origen del link se puede navegar hacia el destino del mismo.

En la figura 4.3 se muestra la especificación formal de la metaclase *Node*, que representa a los nodos del modelo navegacional. Dicha metaclase cuenta con los atributos: *anchors*, representando a los anchors contenidos en el nodo, *modelObjects*, objetos del modelo conceptual observados por el nodo para mostrar información, y *attributes*, datos mostrados por éste en el modelo navegacional. Se define la operación *isNavigableTo* que dice si a partir de ese nodo se puede llegar a otro. Para eso se verifica que ese nodo sea el destino de alguno de sus anchors. Además, se declara una regla de buena formación [1]

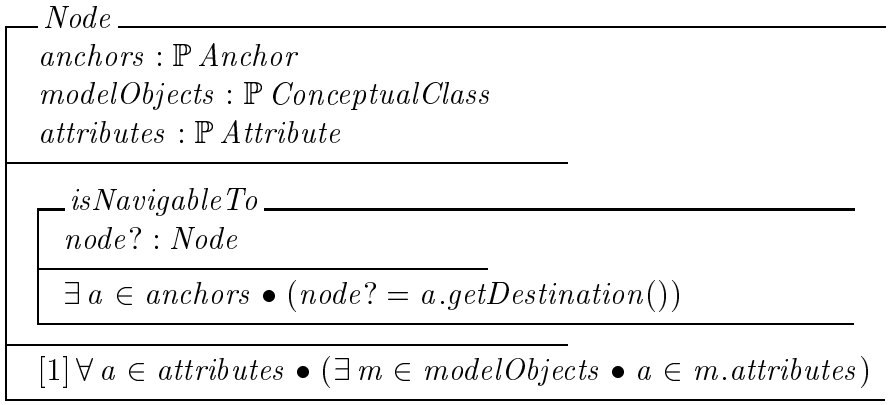

Figura 4.3: Node

que especifica que los atributos del nodo son una proyección de los atributos de los objetos del modelo conceptual del conjunto *modelObjects*. Dado que el nodo es un *observador* (pattern Oberver [GHJV94, p.293]) de tales objetos, se actualiza reflejando cambios en el estado de los mismos.

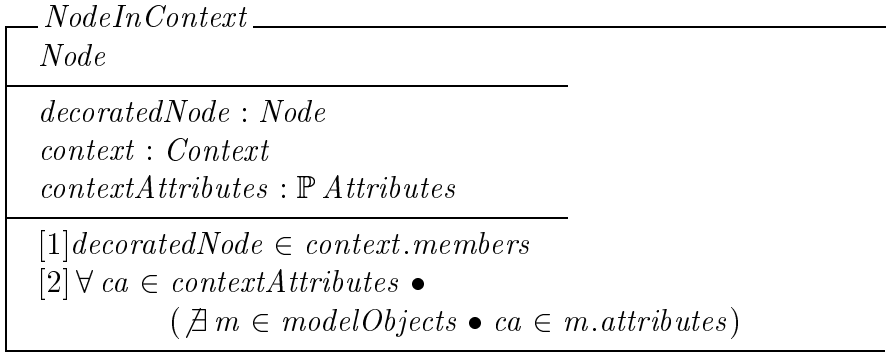

Figura 4.4: NodeInContext

En la figura 4.4 se muestra la metaclase *NodeInContext* que es una de las subclases de *Node*. La misma contiene los atributos: *decoratedNode* que representa al nodo al cual este *NodeInContext decora*, o sea, al cual agrega información del contexto *context*, que es el contexto en el que se encuentra el nodo *decorado*, y *contextAttributes*, los atributos contextuales que el *NodeInContext* agrega al nodo *decorado*. La regla de buena formación [1]

dice que el nodo *decorado* debe pertenecer al contexto del *NodeInContext*, la [2] dice que los atributos presentes en *contextAttributes* son atributos propios del contexto que no pertenecen a ninguna de las clases del modelo conceptual *observadas*. Esto se debe a que el *NodeInContext* agrega al nodo información relacionada con el contexto en que se encuentra.

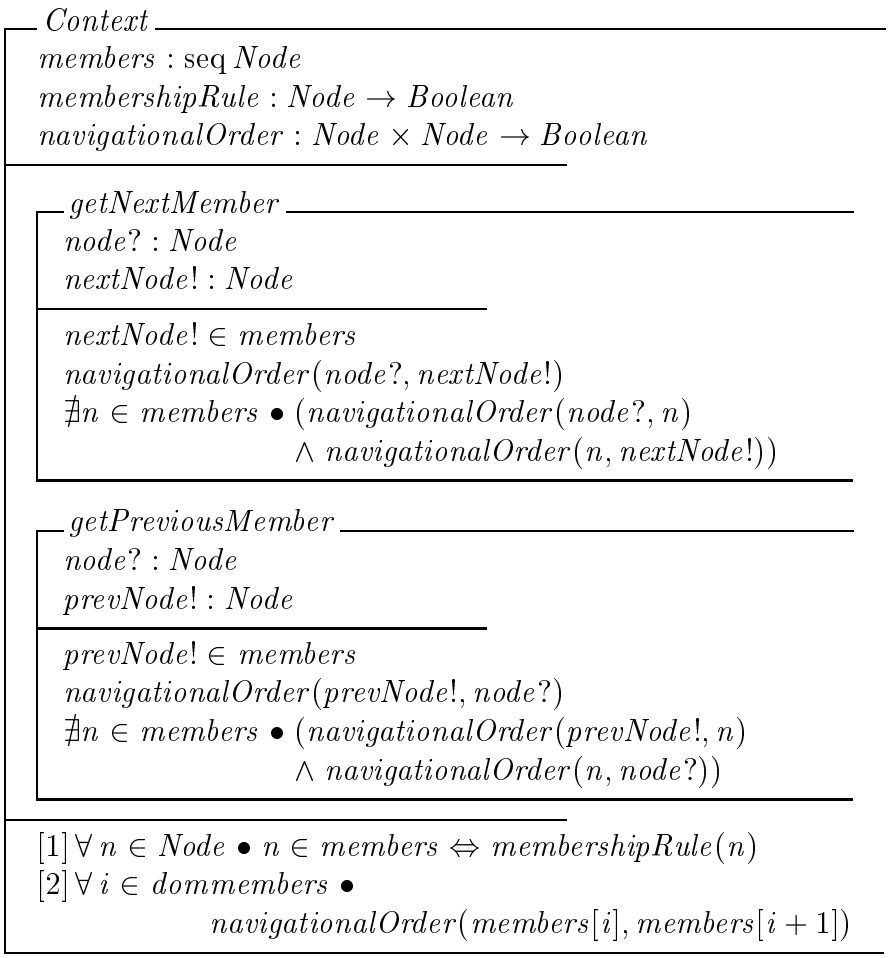

Figura 4.5: Context

En la figura 4.5 se muestra la especificación de la metaclase *Context*, la cual representa a un contexto de navegación del modelo navegacional. Dicha metaclase posee los siguientes atributos: *members*, el conjunto de nodos que conforman el contexto, *membershipRule*, un predicado de pertenencia que deben cumplir todos los miembros del contexto y *navigationalOrder*, un predicado que determina el orden de navegación entre los nodos del contexto.

Ambos predicados pueden ser especificados por extensión (enumeración de los nodos), o por comprensión (mediante una fórmula lógica). Además, se declaran las operaciones *getNextMember* y *getPreviousMember* que retornan respectivamente el próximo y anterior miembro del contexto con respecto a un nodo en particular en el orden de navegación *navigationalOrder*. Por último, se definen dos reglas de buena formación: la [1] declara que un nodo es miembro del contexto si y sólo si cumple con el predicado de pertenencia *membershipRule* (esta regla asegura corrección y completitud del contexto), y la regla [2] declara que entre todos los nodos del contexto se cumple el orden de navegación dado por el predicado *navigationalOrder*.

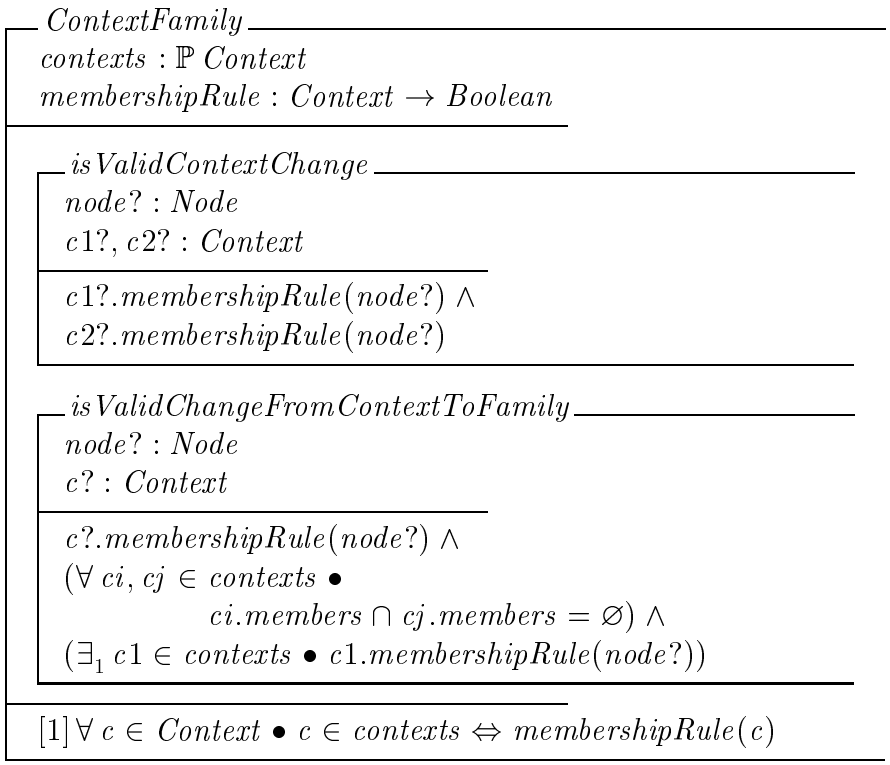

Figura 4.6: ContextFamily

En la figura 4.6 se muestra la especificación en Object-Z de la metaclase *ContextFamily*, que representa a una familia de contextos, la cual posee los atributos *contexts* representando a los contextos que la conforman y *membershipRule*, predicado de pertenencia para los mismos. Se definen dos operaciones: *isValidContextChange* y *isValidChangeFromContextToFamily*. La primera dice que un nodo puede cambiar de un contexto a otro si y sólo

si pertenece a ambos y la segunda determina que un nodo puede cambiar de un contexto a una familia de contextos si y sólo si el nodo pertenece al contexto origen y la familia destino es una partición para ese nodo (los contextos son disjuntos y existe sólo uno que contiene al nodo), de otra forma no se podría determinar a qué contexto de la familia cambiaría el nodo. Por último, se declara la regla de buena formación [1] que establece que los contextos contenidos en la familia deben cumplir con el predicado de pertenencia *membershipRule*.

4.2 Modelo Colaborativo

En esta sección se verán algunas de las metaclases del modelo colaborativo de CHDM especificadas con el lenguaje formal Object-Z [DKRS91]. Tal modelo fue presentado en la sección 3.4 junto con un diagrama de objetos correspondiente. Se mostrarán las especificaciones de las metaclases *ColaborativeModel*, *CollaborativeNode*, *GuidedCouplingStrategy*, *Session* y *AwarenessManager* que han sido seleccionadas por considerarse las más significativas del modelo colaborativo.

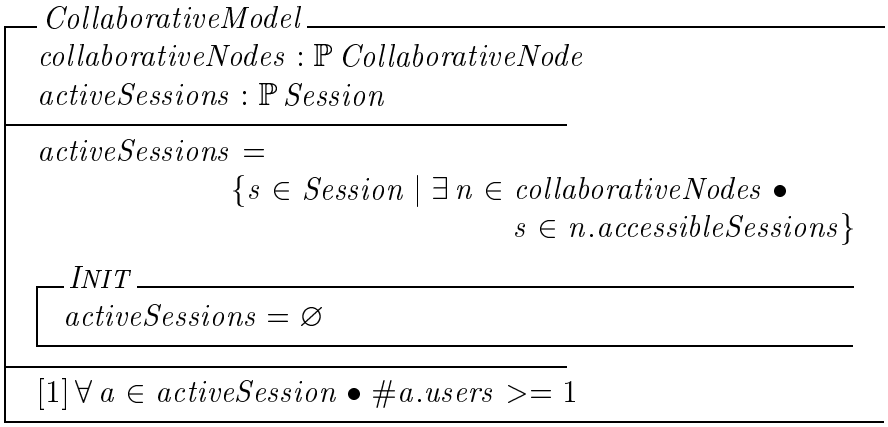

Figura 4.7: CollaborativeModel

En la figura 4.7 se muestra el esquema de Object-Z que especifica a la metaclase *CollaborativeModel*. La misma representa a los modelos colaborativos de CHDM abstrayendo las características y comportamiento comunes a ellos. Cuenta con el atributo *collaborativeNodes*, los nodos colaborativos definidos en el modelo y el atributo derivado *activeSessions* que representa a las sesiones activas en la aplicación. El mismo se calcula pidiéndole las

sesiones accesibles a todos los nodos colaborativos de la aplicación. Además, se ha especificado la operación *INIT* responsable de la inicialización de las sesiones activas como un conjunto vacío, dado que apenas se crea la aplicación, la misma no posee ninguna sesión de colaboración activa. Luego, se define la regla de buena formación [1] que establece que en todas las sesiones activas debe haber por lo menos un usuario, de otra forma la existencia de la sesión no tendría sentido.

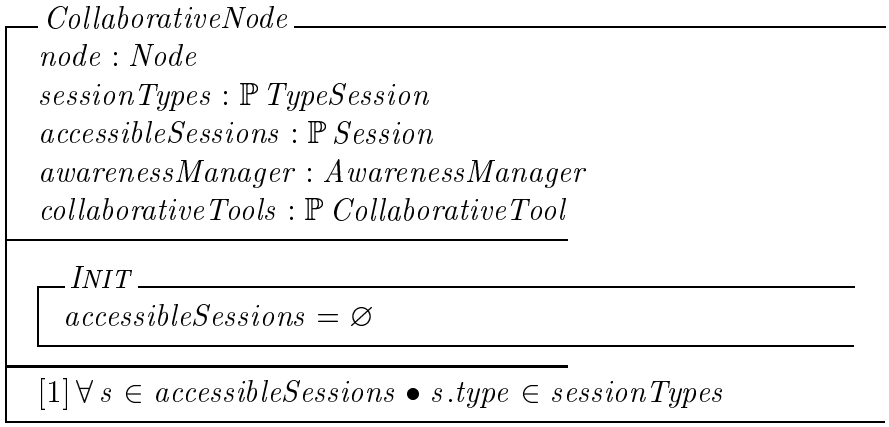

Figura 4.8: CollaborativeNode

En la figura 4.8 se especifica la metaclase que representa a los nodos colaborativos: *CollaborativeNode*. El mismo es un *decorador* [GHJV94, p.175] de un nodo del modelo navegacional, *node*, agregando cualidades colaborativas tales como awareness, herramientas colaborativas, sesiones de colaboración entre usuarios, etc. Los atributos de esta metaclase son *sessionTypes*, un conjunto con los tipos de sesión que el nodo ofrece para que un usuario pueda iniciar, *accessibleSessions*, un conjunto con las sesiones activas accesibles mediante ese nodo, y el *awarenessManager*, encargado de actualizar al nodo con información de *awareness*. El *awarenessManager* lo notifica cada vez que sucede un evento que requiera al nodo actualizar su información de *awareness*. Además, se ha especificado la operación *INIT* responsable de la inicialización de las sesiones accesibles como un conjunto vacío. Esta metaclase también cuenta con la regla de buena formación [1] que especifica que todas las sesiones accesibles desde el nodo colaborativo sean consistentes con los tipos de sesión que dicho nodo ofrece. Es decir, cada sesión accesible desde el nodo debe tener un tipo de sesión habilitada por dicho nodo.

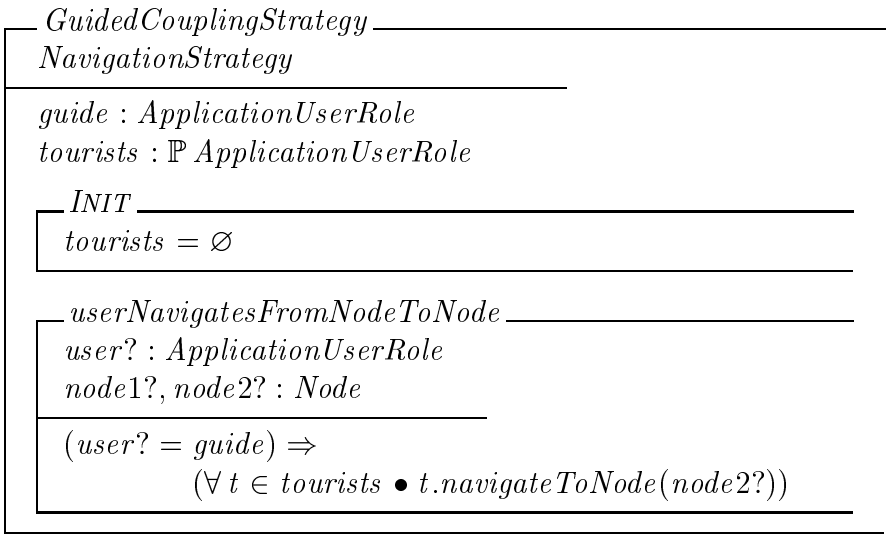

Figura 4.9: GuidedCouplingStrategy

En la figura 4.9 se muestra la metaclase *GuidedCouplingStrategy*, subclase de *NavigationStrategy* que representa a una estrategia de navegación guiada. En este tipo de estrategia de navegación, un usuario guía a un grupo de usuarios, o sea, cada vez que el guía navega hacia un nodo, el resto navega con él. Los atributos de esta clase son *guide*, el usuario guía, y *tourists*, los usuarios que siguen al guía. Además, se ha especificado la operación *INIT* responsable de la inicialización del atributo *tourists* como un conjunto vacío. También se define la operación *userNavigatesFromNodeToNode* que verifica que el usuario que navegó sea el guía, en cuyo caso avisa al resto de los usuarios que deben navegar hacia el mismo nodo.

En la figura 4.10 se muestra la formalización de la clase *Session*, que representa una sesión de trabajo en la cual interviene un conjunto de usuarios. Esta metaclase posee los atributos *users*, los usuarios de la sesión, *couplingStrategy*, la estrategia de acoplamiento en la navegación de dichos usuarios y *type*, el tipo de sesión a la que pertenece. Se ha especificado la operación *INIT* responsable de inicializar la variable *users* como un conjunto vacío. Además se han especificado dos operaciones: *userIn* y *userOut*, para representar la entrada y salida, respectivamente, de un usuario a la sesión. A continuación de las operaciones mencionadas, la regla de buena formación [1] especifica que si la sesión posee una estrategia de navegación *LooselyCoupledStrategy*, la cantidad de usuarios de dicha sesión debe ser

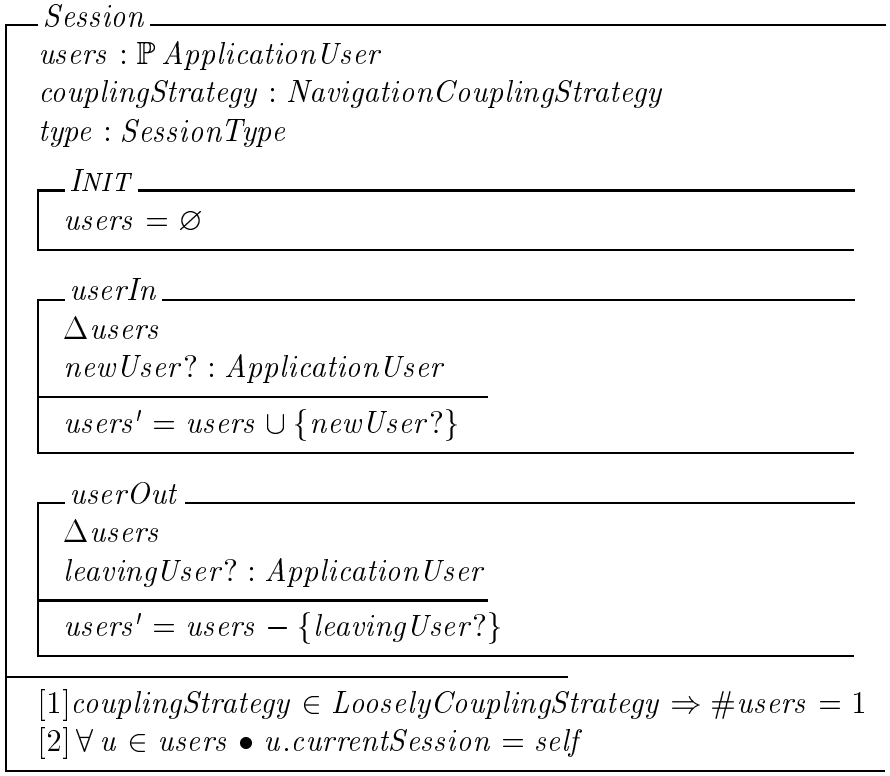

Figura 4.10: Session

necesariamente unitaria, puesto que cada usuario se encontrará navegando independientemente. Es por eso que no tiene sentido considerar más de un usuario en dicha sesión. Por otro lado, el invariante [2] dice que los usuarios de la sesión deben conocerla como su sesión actual *currentSession*.

En la figura 4.11 se muestra la especificación formal de la metaclase *AwarenessManager* que es la encargada de mantener actualizados a los nodos colaborativos con la información de *awareness*. La misma cuenta con el atributo *nodes* que representa al conjunto de nodos dependientes de los eventos de *awareness* a los cuales el *AwarenessManager* debe notificar. La operación de inicialización *INIT* es la encargada de inicializar tal variable como un conjunto vacío. Además, a manera de ejemplo, se ha especificado la operación *userInNode*, la cual crea una notificación de *awareness* y la envía a todos los nodos para que tengan en cuenta el evento de que un usuario ha entrado en un nodo colaborativo en particular, y actualicen su información de *awareness* como corresponda. El *AwarenessManager* también debe

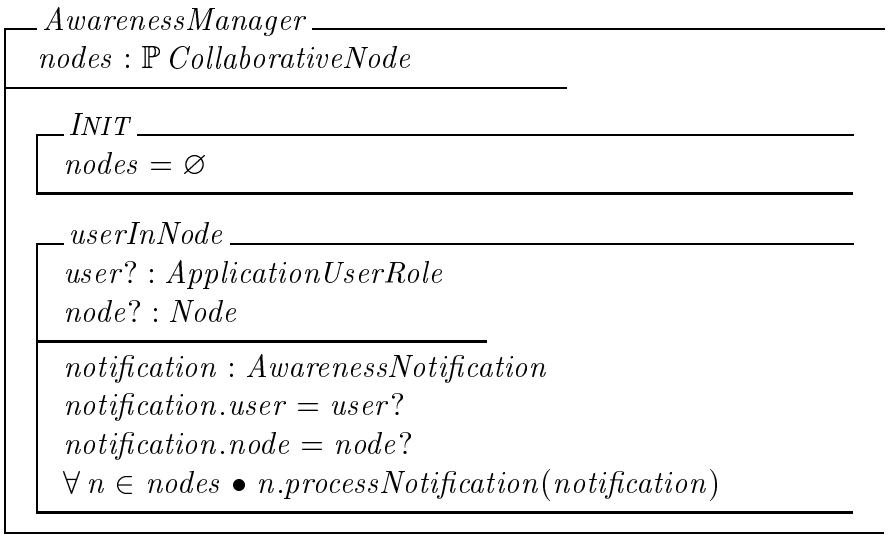

Figura 4.11: AwarenessManager

ser capaz de notificar a los nodos de otros eventos relevantes tales como las acciones que los usuarios se encuentran realizando en la aplicación.

4.3 Ejemplo

En esta sección se mostrará la instanciación de las especificaciones formales de las secciones anteriores para un modelo concreto a fin de mostrar el poder expresivo de la especificación formal. Además se dará, a modo de ejemplo, la demostración de uno de los invariantes especificados en una metaclase para una clase del modelo instanciada a partir de ella.

El ejemplo utilizado fue presentado previamente en la sección 3.4.3, y se muestra su diseño en la figura 3.9. Se trata de una aplicación de hipermedia colaborativa de un laboratorio de investigación. Como fue mencionado en tal sección, el dominio de la aplicación consiste de miembros del laboratorio, áreas y proyectos de investigación. Se modelará el nodo del mapa de la aplicación a partir del cual se puede establecer un tour guiado por la misma.

Como se explicó en el capítulo 3, para definir el diseño de una aplicación con la metodología CHDM es necesario instanciar cada uno de los tres modelos (conceptual, navegacional y colaborativo) con las clases específicas de la aplicación. A continuación se mostrarán cada uno de los modelos instanciados para el ejemplo del laboratorio de investigación.

4.3.1 Modelo Conceptual

Para instanciar el modelo conceptual de una aplicación específica, es necesario instanciar el modelo conceptual de CHDM (ver sección 3.2). Se deben declarar las clases que intervienen en el modelo elegido como instancias de las metaclases provistas por CHDM. Para eso, se deben identificar los objetos relevantes al dominio de la aplicación.

A continuación se muestra la declaración de las clases intervinientes en el modelo concepual:

$$LabMember, ResearchArea, Project : Class;$$
$$LabMemberName, ResearchAreaName, ProjectName,$$
$$Telephone, Address, Responsible, Members, Leader : Attribute;$$

Se cuenta con *LabMember*, *ResearchArea* y *Project*, instancias de *Class* y las clases *LabMemberName*, *ResearchAreaName*, *ProjectName*, *Telephone*, *Address*, *Responsible*, *Members* y *Leader*, instancias de *Attribute*.

Class y *Attribute* son metaclases definidas en el modelo conceptual de CHDM.

$$\begin{array}{|l} LabMemberName : Attribute \\ \hline perspectives = \{String\} \end{array}$$

Figura 4.12: Name

En la figura 4.12 se muestra el esquema en Object-Z para la instanciación de la clase *LabMemberName* como instancia de *Attribute*. La misma cuenta con el atributo *perspectives* que representa a los tipos o clases a las que puede pertenecer. Recordar que OOHDM permite que los atributos sean multitipados (característica heredada por CHDM). En este caso el conjunto es unitario y contiene a la clase *String* puesto que el atributo *LabMemberName* tiene sólo un tipo. Dicha clase será utilizada para definir uno de los atributos de la clase *LabMember* que se muestra en la figura 4.13.

Por su parte, la clase *LabMember* (instancia de *Class*) cuenta con el atributo *attributes* que consiste de un conjunto con diferentes objetos tales como *LabMemberName*, *Telephone*, *Address* y otros que no se han considerado en este ejemplo por cuestiones de simplicidad. Esto significa que a la hora de instanciar un miembro concreto a partir de la clase *LabMember*, digamos *Mary*, contará con atributos de tales clases.

$LabMember : Class$
$attributes = \{LabMemberName, Telephone, Address, ...\}$

Figura 4.13: LabMember

$ResearchArea : Class$
$attributes = \{ResearchAreaName, Responsible, LabMember, ...\}$

Figura 4.14: ResearchArea

En la figura 4.14 se muestra el esquema de la clase *ResearchArea*, la cual también cuenta con el atributo *attributes* (por ser también instancia de la metaclase *Class*), pero en este caso contiene los objetos de las clases *ResearchAreaName*, *Responsible* y *LabMember*. Esto significa, como en el caso anterior, que a la hora de instanciar un área de investigación concreta, contará con atributos de tales clases.

$Project : Class$
$attributes = \{ProjectName, Leader, LabMember, ...\}$

Figura 4.15: Project

Por último, en la figura 4.15 se muestra el esquema de la instanciación de la clase *Project*, también instancia de *Class*. En este caso los elementos que conforman el conjunto de atributos *attributes* son *ProjectName*, *Leader* y *LabMember*, de tal forma que un proyecto en particular contará con atributos de tales clases.

4.3.2 Modelo Navegacional

En esta sección se instanciará el modelo navegacional de CHDM (ver figura 3.3.2) para el ejemplo que se está desarrollando. Para ello es necesario identificar qué nodos y links van a conformar el espacio de navegación de la aplicación.

Para el modelo navegacional del ejemplo sólo se definirá, por cuestiones de simplicidad, la clase de nodo *SiteMapNode*, instancia de *Node*, que representa el nodo que muestra el mapa del sitio y tres clases de anchors que el nodo contendrá: *LabMemberAnchor*, *ResearchAreaAnchor* y *ProjectAnchor*. Notar que en la aplicación sólo se definirá una instancia de la clase *SiteMapNode*: el nodo del mapa del sitio.

A continuación se muestra la definición como instancias de las clases del modelo navagacional.

$SiteMapNode : Node;$
$LabMemberAnchor, ResearchAreaAnchor, ProjectAnchor : Anchor;$

$SiteMapNode : Node$
$modelObjects = \{LabMember, ResearchArea, Project\}$ $attributes = \{LabMemberName, ResearchAreaName,$ $\quad ProjectName\}$ $anchors = \{LabMemberAnchor, ResearchAreaAnchor,$ $\quad ProjectAnchor\}$

Figura 4.16: siteMapNode

En la figura 4.16 se muestra el esquema de instanciación correspondiente a la clase *SiteMapNode*. La misma cuenta con los atributos definidos en su metaclase: *modelObjects*, *attributes* y *anchors* (ver figura 4.3). Los *modelObjects* son las clases *LabMember*, *ResearchArea* y *Project*, o sea, el nodo observa miembros, áreas y proyectos del modelo conceptual. La variable *attributes* consiste de *LabMemberName*, *ResearchAreaName* y *ProjectName* lo que significa que el nodo muestra tales atributos de los objetos que observa. Por último, la variable *anchors* consiste de tres posibles tipos de anchors: *LabMemberAnchor*, *ResearchAreaAnchor* y *ProjectAnchor*, o sea el nodo tiene anchors a miembros, áreas y proyectos del laboratorio.

Un Ejemplo de Verificación Formal

Una de las ventajas mencionadas de haber provisto una base formal para la metodología de diseño CHDM, es la posibilidad de realizar verificaciones de invariantes definidos formalmente para las metaclases. Las clases de los modelos instanciados a partir de las diferentes etapas de CHDM cumplirán

con tales invariantes, justamente por tratarse de instancias de las metaclases de dichos metamodelos.

A continuación se muestra un ejemplo de una verificación formal; en este caso se demostrará la regla [1] de la metaclase *Node* especificada con Object-Z en la figura 4.3 para el nodo *SiteMapNode*. Dicho invariante establece que los atributos del nodo son una proyección de los atributos de los objetos del modelo conceptual observados por dicho nodo. O lo que es lo mismo, el nodo no puede agregar atributos por sí mismo. De la misma manera se podría verificar formalmente cualquiera de los invariantes definidos para las metaclases de CHDM, de cualquiera de las etapas del mismo. Una consecuencia importante de este hecho es que se ve conformada una base formal para la metodología y si, por ejemplo, se construyera un editor para CHDM, el mismo podría verificar automáticamente las reglas de buena formación provistas, asegurando así que se cumplieran para los modelos diseñados mediante dicha herramienta de edición.

Propiedad 4.3.1 *El invariante*
$\forall\, n \in Node \bullet \forall\, a \in attributes \bullet \exists\, m \in modelObjects \bullet a \in m.attributes$
definido en Node es válido para su instancia SiteMapNode.

Demostración: Sea $a \in SiteMapNode.attributes$.
Por definición de $SiteMapNode.attributes$,
$a \in \{LabMemberName, ResearchAreaName, ProjectName\}$
Si $a = LabMemberName$, por definición de $LabMember$,
$a \in LabMember.attributes$
Además, por la definición de $SiteMapNode$, se sabe que
$LabMember \in SiteMapNode.modelObjects$.

$\therefore\, \exists\, LabMember \in SiteMapNode.modelObjects \bullet a \in LabMember.atrtributes$
Análogamente, se cumple para $a = ResearchAreaName$ y $a = ProjectName$.

Queda así demostrada la propiedad 4.3.1 $\square$

4.3.3 Modelo Colaborativo

Para definir el modelo colaborativo de una aplicación específica, es necesario instanciar el modelo colaborativo de CHDM (ver sección 3.4). Para eso, primero se deben elegir los nodos a los que se les agregará alguna funcionalidad colaborativa y luego definir un tipo de nodo colaborativo para cada uno.

En el caso del ejemplo del laboratorio de investigación, se ha elegido el nodo del mapa del sitio para agregarle la funcionalidad de una navegación

guiada por la aplicación. En la misma, un usuario miembro del laboratorio guía a un grupo de usuarios externos por la aplicación. Cuando el guía navega, los demás navegan con él.

A continuación se muestra la definición de las clases del modelo colaborativo necesarias para la implementación del tour guiado.

TourSession : *Session*
TourSessionType : *SessionType*
CollaborativeSiteMapNode : *CollaborativeNode*
LabNavigationalStrategy : *GuidedCouplingStrategy*
LabUser : *ApplicationUser*
LabMemberRole, *NonLabMemberRole* : *AplicationUserRole*
PortalTourGuideModel : *CollaborativeModel*

LabNavigationalStrategy : *NavigationalCouplingStrategy*
guide = *LabMemberRole* *tourists* = {*NonLabMemberRole*}

Figura 4.17: LabNavigationalStrategy

En la figura 4.17 se define la instanciación de la clase encargada de la estrategia de navegación guiada, *LabNavigationalStrategy*. La misma especifica que el usuario guía debe tener el rol de *LabMemberRole*, o sea, ser miembro del laboratorio y que los "turistas" deben pertenecer a la clase *NonLabMemberRole*, o sea, ser externos al laboratorio.

CollaborativeSiteMapNode : *CollaborativeNode*
node : *SiteMapNode* *sessionTypes* = {*TourSessionType*} *collaborativeTools* = ∅ *accessibleSessions* = {*TourSession*}

Figura 4.18: CollaborativeSiteMapNode

En la figura 4.18 se muestra el esquema de instanciación de la clase *CollaborativeSiteMapNode*. Dicha clase tendrá una sola instancia que representará al nodo colaborativo del mapa del sitio (observador del nodo del mapa

del sitio del modelo navegacional). De esta forma, el atributo *node* se define como el anteriormente definido *SiteMapNode*. Los tipos de sesión que este nodo colaborativo ofrece quedan especificadas en el atributo *sessionTypes*, definido como el conjunto unitario que contiene a *SessionType*, representando que el nodo colaborativo tiene un conjunto de tipos de sesión que pueden ser iniciadas a partir de él. Por otro lado, se especifica que en este nodo colaborativo no existen herramientas de colaboración definiendo el atributo *collaborativeTools* como el conjunto vacío, y por último las sesiones accesibles a través de este nodo quedan descriptas en el atributo *accessibleSessions*, definido como el conjunto unitario que contiene a la clase *TourSession*, que representa a una sesión de tour, la cual tiene asociada una estrategia de navegación guiada *LabNavigationalStrategy*.

$LabTourGuideModel : CollaborativeModel$
$collaborativeNodes = \{CollaborativeSiteMapNode, ...\}$

Figura 4.19: LabTourGuideModel

Para tener una visión general del modelo colaborativo se define el modelo *LabTourGuideModel* (figura 4.19) como instancia de *CollaborativeModel* y se define el conjunto de los nodos colaborativos intervinientes. En este caso, basta incluir el anteriormente definido *CollaborativeSiteMapNode* (figura 4.18) más todos los nodos que formen parte de la aplicación.

Capítulo 5

Ejemplos

En este capítulo se describirán dos ejemplos de diseños modelados con la metodología CHDM propuesta en este trabajo de grado para dos aplicaciones de hipermedia colaborativas. El primero de ellos, *Portinari colaborativo*, se basa en la aplicación del Museo de Portinari [TS] previamente diseñada con la metodología OOHDM [Ros96]. Dicha aplicación hipermedial consta de información sobre el artista plástico Candido Portinari y sus obras de arte. Se mostrarán diferentes características colaborativas que CHDM permite modelar, tales como *awareness*, herramientas colaborativas, sesiones de colaboración, roles de usuarios y navegación acoplada. El segundo ejemplo, *DOLPHIN*, describe una aplicación colaborativa desarrollada por el laboratorio GDM-IPSI [GMD] que provee asistencia por computadora a distintos tipos de reuniones de usuarios, tanto en espacios físicos como virtuales.

En la sección 5.1 se definirá una posible implementación del mecanismo de *awareness* dentro de una aplicación diseñada con CHDM. Luego, en las secciones 5.2.3 y 5.3.3 se mostrarán los ejemplos de la aplicación de CHDM al Museo de Portinari y a la aplicación DOLPHIN, respectivamente. Por último, en la sección 5.4 se comentarán los resultados obtenidos.

5.1 Implementación de *Awareness*

En esta sección se presentará una posible implementación del mecanismo de *awareness* para una aplicación de hipermedia colaborativa modelada con CHDM, que será utilizada para modelar los ejemplos de las secciones siguientes. La misma consiste en habilitar a los nodos colaborativos para recibir notificaciones de los tipos de *awareness* que quieran implementar. Para ello cada nodo deberá implementar ciertas interfases de acuerdo a los diferentes tipos de *awareness*. Por otro lado, se cuenta con los objetos *AwarenessNotification*

que modelan las notificaciones de *awareness* que puede recibir un nodo colaborativo. Dichos objetos son creados por el *AwarenessManager* en el momento en que tales eventos suceden, y enviados al nodo correspondiente mediante el mensaje *processNotification*(*AwarenessNotification*). El nodo le enviará a la notificación el mensaje *notify*(*self*) para indicarle que puede recibir el evento de *awareness* que ella modela, que dependerá del tipo de notificación del que se trate. Así el nodo reaccionará de acuerdo al mensaje recibido, actualizando su información de *awareness*.

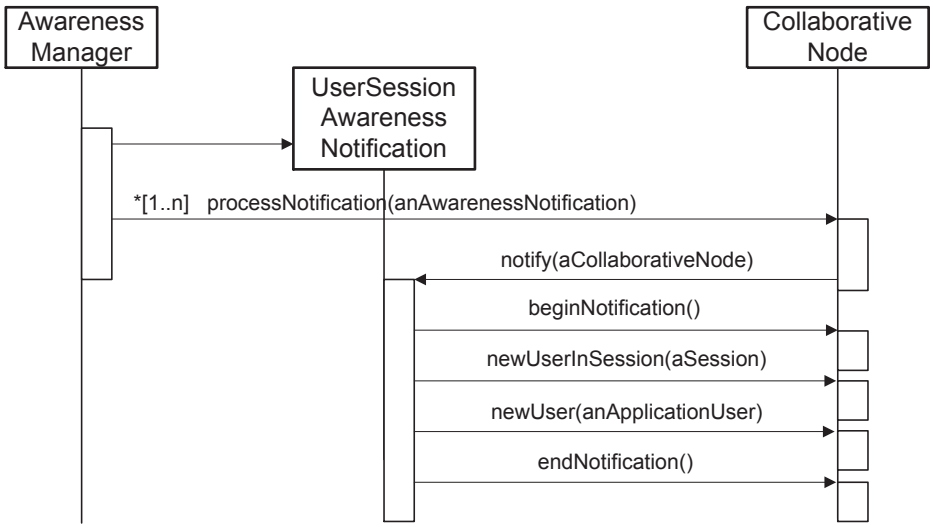

Figura 5.1: Notificación de *Awareness* Compuesta

En la figura 5.1 se muestra la secuencia de acciones correspondiente a la llegada de una notificación de *awareness* compuesta a un nodo colaborativo. Tal notificación está compuesta por información de usuario y de sesión, por lo tanto la notificación le envía mensajes correspondientes a los dos eventos.

A continuación se describirá el mecanismo en detalle. Primero es necesario identificar los tipos de *awareness* básicos necesarios para la aplicación. Para este ejemplo se utilizarán los siguientes tipos: *awareness* de usuario, lugar, acción y objeto. A partir de tales componentes básicos se pueden construir composiciones para representar tipos de *awareness* más complejos. Por ejemplo, para definir el tipo de *awareness* de presencia (que indica en qué lugar está un usuario) será necesario componer los tipos de *awareness* básicos de usuario y lugar.

En la figura 5.2 se muestra la jerarquía conceptual propuesta de tipos de *awareness*. Dicha jerarquía se encuentra encabezada por la clase *AwarenessComponent* la cual es abstracta, con las subclases *UserAwareness*,

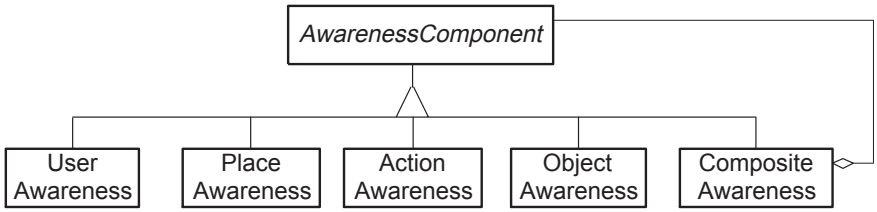

Figura 5.2: Jerarquía de Tipos de *awareness*

awareness de usuario, *PlaceAwareness*, de lugar, *ActionAwareness*, de acción, *ObjectAwareness*, de objeto y la clase *CompositeAwareness* representando a los tipos de *awareness* compuestos.

Una vez definidos los tipos de *awareness* básicos es necesario construir dos jerarquías de objetos que deberán ser implementadas. La primera, una jerarquía de interfases con mensajes correspondientes a eventos relacionados a cada uno de los tipos de *awareness* básicos.

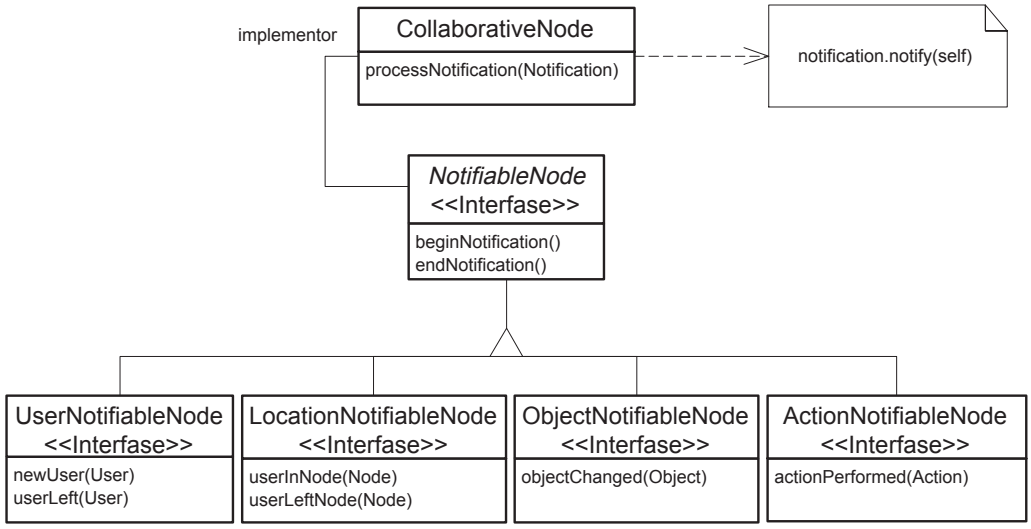

Figura 5.3: Jerarquía de Interfase NotifiableNode

En la figura 5.3 se muestra la jerarquía de *NotifiableNode*, clase abstracta que cuenta con las subclases *UserNotifiableNode*, nodo notificable de *awareness* de usuario, *LocationNotifiableNode* de lugar, *ObjectNotifiableNode* de objeto y *ActionNotifiableNode*, nodo notificable de *awareness* de acción. Tales interfases serán implementadas por el nodo colaborativo de acuerdo al

tipo de *awareness* que provea. Implementar una de estas interfases lo habilita para recibir notificaciones de los distintos tipos de *awareness*. Para esta jerarquía se utilizó el patrón de diseño orientado a objetos *Composite* [GHJV94, p.163]. Por ejemplo, un nodo colaborativo que implemente la interfase *UserNotifiableNode* puede recibir los mensajes *userIn*(*User*) o *userLeft*(*User*), que indican que un usuario llegó o dejó un nodo de la aplicación, respectivamente. Para los casos de tipos de *awareness* compuestas, el nodo colaborativo deberá implementar las interfases correspondientes a cada una de sus componentes.

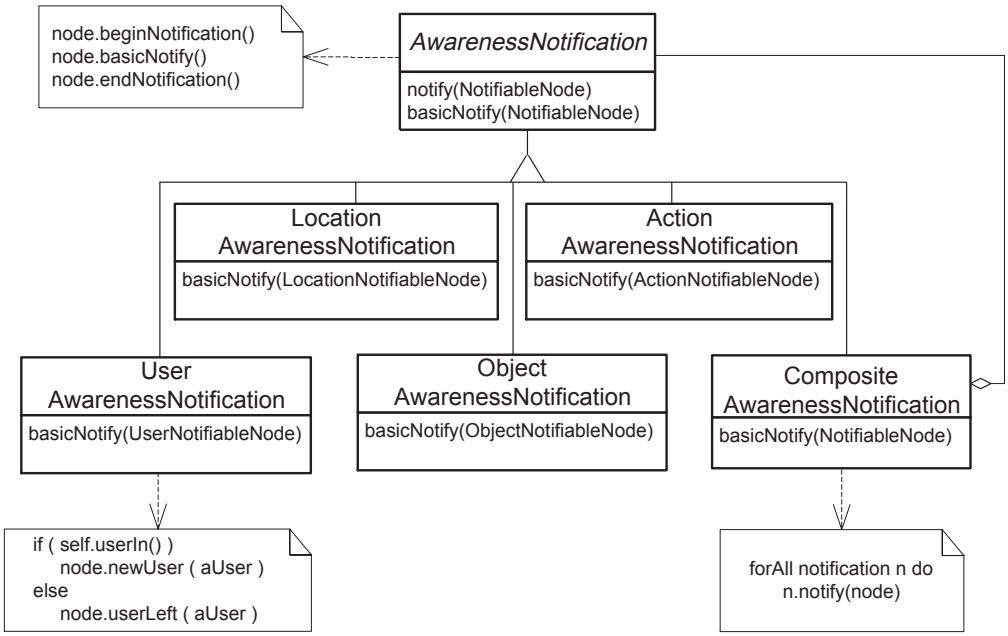

Figura 5.4: Jerarquía de Notificación de *awareness*

La segunda jerarquía que es necesario implementar se muestra en la figura 5.4 y se trata de los objetos que representan notificaciones de *awareness*. La misma cuenta con la superclase *AwarenessNotification* que es abstracta y cuenta con las subclases *UserAwarenessNotification*, que modela una notificación de *awareness* de usuario, *LocationAwarenessNotification* de lugar, *ObjectAwarenessNotification* de objeto, *ActionAwarenessNotification* de acción y la clase correspondiente a las notificaciones de *awareness* compuestas *CompositeAwarenessNotification*. Para este diseño también se utilizó el patrón *Composite* [GHJV94, p.163]. Las notificaciones son los objetos encargados de notificar a los nodos de los eventos de *awareness* correspondientes. Por ejemplo, una instancia de *UserAwarenessNotification* puede enviar-

le a un nodo que implemente la interfase *UserNotifiableNode* los mensajes *newUser*(*User*) o *userLeft*(*User*).

Como comentario final, es importante destacar que la implementación descripta puede ser utilizada en cualquier aplicación diseñada con CHDM, ya que no depende de ninguna arquitectura ni lenguaje de programación en particular. Además, el hecho de modelar los tipos básicos de *awareness* y utilizar el patrón *Composite* para representar su jerarquía, evita que la cantidad de clases representando a tales objetos crezca exponencialmente de acuerdo a las combinaciones que la aplicación requiera. Fácilmente se pueden agregar nuevos tipos básicos de *awareness* y construir distintos tipos de *awareness* compuestos.

5.2 Portinari colaborativo

En esta sección se describirá un ejemplo de la construcción de una aplicación de hipermedia colaborativa para un museo de arte: "Portinari colaborativo". El mismo cuenta con datos sobre pinturas, pintores, ubicación de las pinturas dentro del museo, salas de arte y entrevistas a los pintores, entre otras cosas. Se mostrarán brevemente los modelos conceptual y navegacional, y se detallará el modelo colaborativo junto con diferentes actividades colaborativas que el mismo permite realizar a los usuarios tales como: navegar mediante un tour guiado por un representante del museo, entablar discusiones de arte con otros usuarios, ver la ubicación de otros usuarios en las salas del museo o navegar de manera independiente.

Es importante mencionar que se ha construido una aplicación de hipermedia colaborativa en base a una aplicación de hipermedia existente. Se trata del Museo de Arte de Portinari [TS] que ha sido diseñado con la metodología OOHDM [Ros96].

En las secciones 5.2.1 y 5.2.2 se mostrarán versiones simplificadas de los modelos conceptual y navegacional existentes (presentados en [Ros96]). Luego, en la sección 5.2.3, se mostrará un posible diseño de modelo colaborativo.

5.2.1 Modelo Conceptual

En esta sección se presentará una versión simplificada del modelo conceptual del Museo de Arte de Portinari [TS]. Como fue mencionado anteriormente, los objetos relevantes del dominio de la aplicación son los pintores, las pinturas, las salas del museo, entrevistas, etc. En la figura 5.5 se muestran las clases que modelan dichos objetos junto con las relaciones existentes entre ellas y las cardinalidades correspondientes.

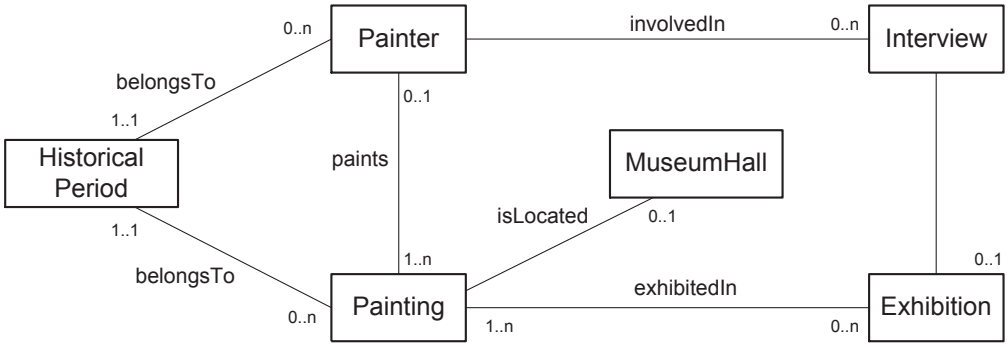

Figura 5.5: Modelo Conceptual del Museo de Arte

Por ejemplo, la relación *isLocated* que se da entre la clase *Painting* y *MuseumHall* representa la ubicación física de las pinturas dentro del museo y tiene cardinalidad 0..1 puesto que una pintura está como máximo en una sala a la vez, y como mínimo en ninguna, pues puede suceder que sea temporariamente prestada a otro museo.

5.2.2 Modelo Navegacional

En esta sección se presenta una versión simplificada del modelo navegacional de la aplicación del Museo de Arte de Portinari presentado en [Ros96]. El mismo se muestra en la figura 5.6.

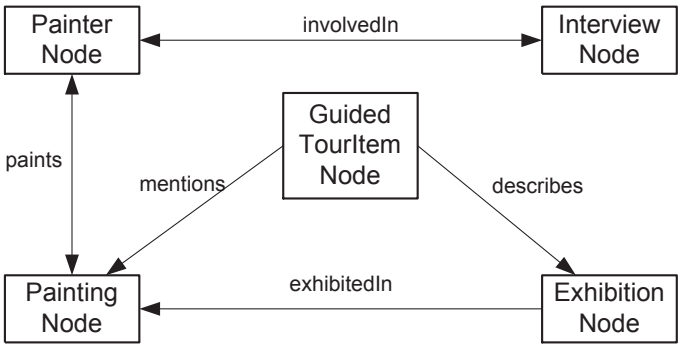

Figura 5.6: Modelo Navegacional del Museo de Arte

El modelo navegacional representa a los nodos de la aplicación junto con los links existentes entre ellos, los cuales permiten realizar la navegación (las direcciones de las flechas indican las direcciones de navegación posibles).

Notar que no todas las clases del modelo conceptual fueron especificadas como nodos, se eligen las más significativas de acuerdo a cómo se quiere construir el espacio de navegación. Por ejemplo, las clases *HistoricalPeriod* y *MuseumHall* no tienen nodos asociados, puesto que no se consideró necesario.

Por otro lado, ha sido agregado el nodo *GuidedTourItemNode* que representa un nodo dentro de un tour guiado, previamente construido para ser recorrido. El mismo permite navegar hacia pinturas o exhibiciones relacionadas con ellas.

5.2.3 Modelo Colaborativo

En esta sección se verán las diferentes características colaborativas agregadas a la aplicación del Museo de Arte de Portinari. Como se dijo anteriormente, a la aplicación se le desean agregar las siguientes características colaborativas: la posibilidad de realizar un tour guiado por la aplicación, sesiones de discusión entre usuarios y *awareness* entre usuarios de la aplicación. El diagrama del modelo colaborativo se verá en dos figuras diferentes para una mejor comprensión. Nótese que el modelo colaborativo que se verá a continuación es instancia del metamodelo colaborativo de CHDM propuesto, por lo tanto cada una de sus clases son *instancias* de las metaclases de CHDM.

En la figura 5.7 se muestra la porción del modelo colaborativo que modela la posibilidad de realizar un tour guiado por la aplicación. Para tal fin, se definió el nodo colaborativo *PainterCollaborativeNode* correspondiente a un pintor, a partir del cual se puede iniciar una sesión de tour guiado. Dicha clase es instancia de la metaclase *CollaborativeNode* del metamodelo colaborativo de CHDM. Para representar a las sesiones de navegación guiada se tiene a la clase *GuidedTourSession*, representando a una sesión de tour, que tiene una estrategia de navegación *GuidedTourCouplingStrategy* (instancia de la metaclase *GuidedCouplingStrategy*), que representa a una estrategia de navegación guiada. En tal tipo de estrategia de navegación un usuario con rol *Guide* guía a los usuarios con rol *Visitor* por el museo. Además, es posible establecer una sesión privada, de navegación independiente, por la aplicación. El nodo mencionado tiene asociado un manejador de *awareness* concreto (instancia de la metaclase *AwarenessManager*) denominado *ArtMuseumAwarenessManager*, encargado de avisarle sobre los eventos de *awareness* que el nodo esté interesado.

Nótese que el modelo colaborativo de la figura 5.7 es muy similar al del ejemplo presentado en la sección 3.4.3 del tour por un laboratorio de investigación. Eso se debe a que todos los modelos colaborativos de CHDM son instancias del metamodelo propuesto por la metodología y así el metamodelo sirve de molde para ellos.

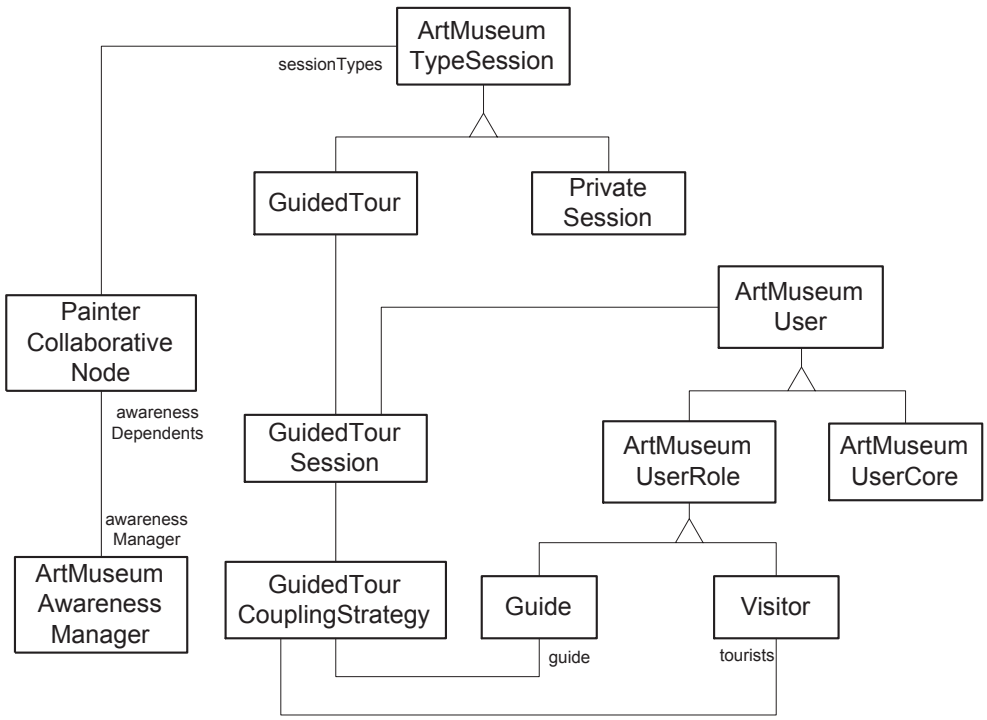

Figura 5.7: Modelo Colaborativo: Tour guiado

El nodo colaborativo *PaintingCollaborativeNode* correspondiente a una pintura se muestra en la figura 5.8. A partir de él se puede establecer una sesión de discusión con otros usuarios, que tiene asociada una estrategia de navegación no acoplada, *LooselyCoupledStrategy*, puesto que los usuarios pueden navegar en forma independiente mientras establecen una discusión. Para proveer la funcionalidad de discusión entre usuarios, el nodo tiene asociada una herramienta sincrónica *DiscussionSynchronicTool*.

Es importante mencionar que el nodo *PaintingCollaborativeNode* tiene asociado el mismo *AwarenessManager* que el nodo *PainterCollaborativeNode* en el escenario del tour anterior, puesto que los diferentes tipos de *awareness* están determinados por las distintas interfases que los nodos colaborativos implementen. Según la implementación de *awareness* propuesta en la sección 5.1 los nodos colaborativos implementan una interfase de notificaciones de *awareness* en particular y se "suscriben" al *AwarenessManager* para que éste los notifique de los eventos correspondientes.

El nodo colaborativo *MuseumMapCollaborativeNode* se muestra en la figura 5.9. El mismo implementa la interfase *LocationNotifiableNode*, pues-

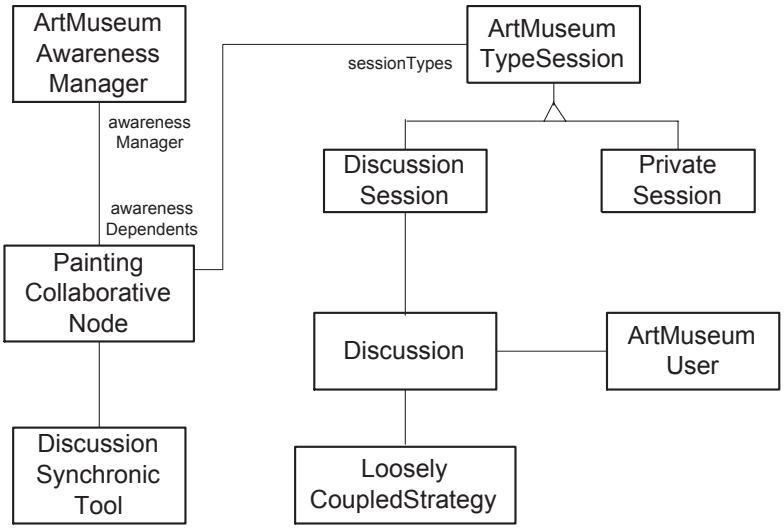

Figura 5.8: Modelo Colaborativo: Sesión de Discusión

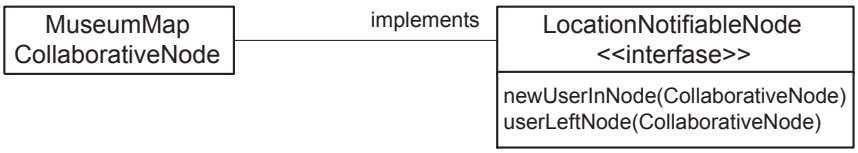

Figura 5.9: Implementación de una Interfase de *Awareness*

to que el tipo de *awareness* que provee es de *location*, para poder especificar en qué salas hay usuarios. Por esta razón responde a los mensajes *newUserInNode*(*CollaborativeNode*) y *userLeftNode*(*CollaborativeNode*), invocados por el *AwarenessManager* para indicarle al nodo que un usuario llegó a él o navegó hacia otro nodo.

Por último, en la figura 5.10 se muestra una notificación de *awareness* de *location*, *LocationAwarenessNotification*, la cual implementa el mensaje *basicNotify* (heredado de su superclase *AwarenessNotification*), enviándole al nodo correspondiente el mensaje *newUserInNode*(*CollaborativeNode*).

5.3 DOLPHIN

DOLPHIN es una aplicación de hipermedia colaborativa desarrollada por el laboratorio GMD-IPSI para soportar la preparación y manejo de reuniones grupales en diferentes escenarios: reuniones "cara a cara" así como también

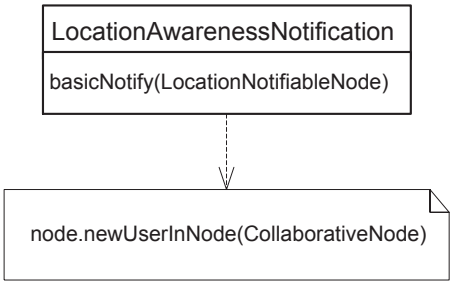

Figura 5.10: Implementación de una Notificación de *Awareness*

reuniones con participantes remotos. Las actividades que deben soportarse en este tipo de escenarios incluyen manejo de reuniones, sesiones de *brainstorming*, organización de ideas, discusión y toma de decisiones. Muchas de estas actividades están basadas en documentos que son preparados antes de la reunión (por ejemplo una agenda, lista de tópicos, propuestas, gráficos), presentados y editados durante la reunión, y finalmente pueden obtenerse nuevos documentos que resumen las ideas tratadas en la misma. Este material muchas veces se procesa luego de la reunión y pasa a ser el punto de partida para reuniones siguientes.

Combinando conceptos de las áreas de hipermedia y ambientes colaborativos, DOLPHIN provee funcionalidad para manejar estos documentos ya sea en un entorno compartido o privado. La manipulación de estos documentos hipermediales incluye tanto el agregado de páginas y links a la estructura del documento, así como también el agregado de contenido propiamente dicho en cualquiera de las páginas. A su vez se permite que los participantes trabajen solos o fuertemente acoplados al grupo, en un ambiente compartido.

En las secciones 5.3.1, 5.3.2 y 5.3.3 se comentarán versiones simplificadas de los modelos conceptual, navegacional y colaborativo de DOLPHIN.

5.3.1 Modelo Conceptual

Se ha elaborado un modelo conceptual reducido que captura las principales entidades de la aplicación DOLPHIN, figura 5.11. En la misma, dado que lo más importante que se manipula es un documento hipermedial, es preciso modelarlo junto con los posibles elementos que lo conforman. Estos últimos se encuentran modelados a través de la jerarquía *DocumentContent* y son *Page*, que representa a una página dentro de un documento, *Link*, un link que relaciona páginas, *Scribble*, un "garabato", *Image*, una imagen y *Text*, un texto. Aquí se hace uso del patrón de diseño *Composite* [GHJV94, p.163],

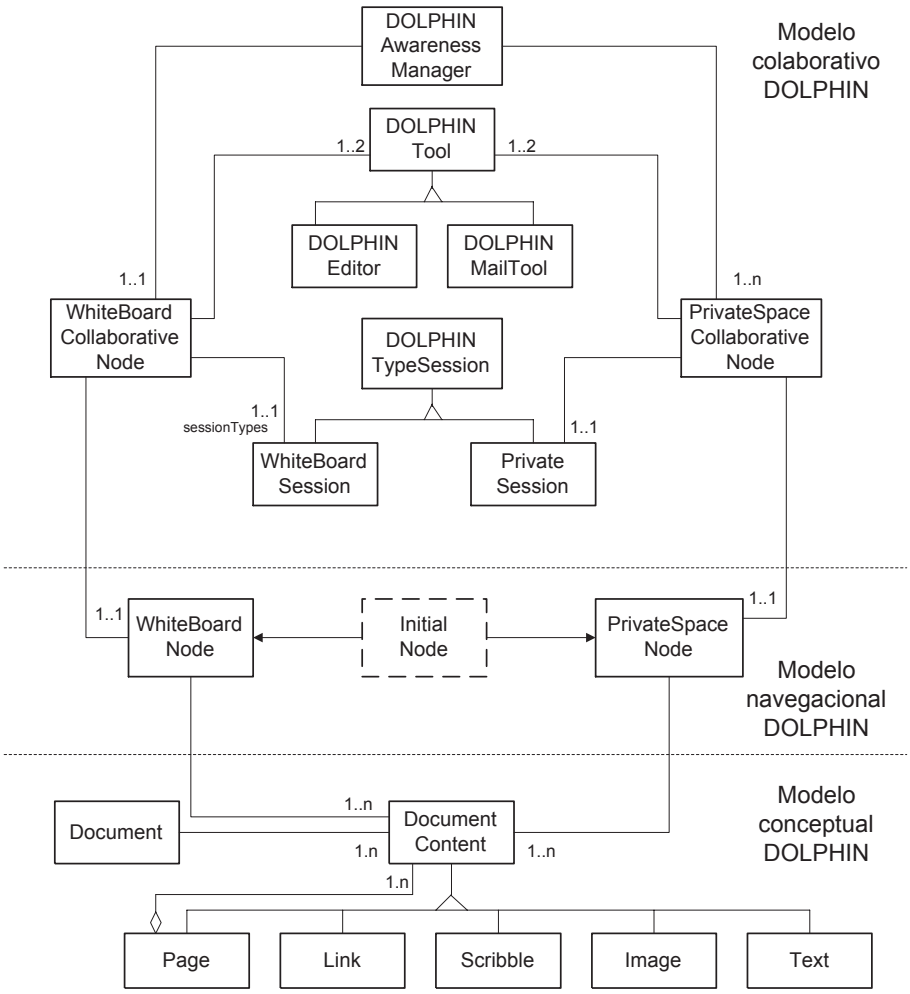

Figura 5.11: Modelos de DOLPHIN

ya que una página es un posible contenido de documento que a su vez está compuesto por otros elementos de contenido.

5.3.2 Modelo Navegacional

El modelo navegacional reducido planteado para DOLPHIN se circunscribe básicamente a un índice (rectángulo de línea cortada) desde el cual se puede acceder al entorno grupal o al individual y dos nodos principales que representan justamente estas dos últimas alternativas: *WhiteBoardNode* y *PrivateSpaceNode*, figura 5.11. En el primero de ellos, los participantes de la reunión pueden editar el documento compartido.

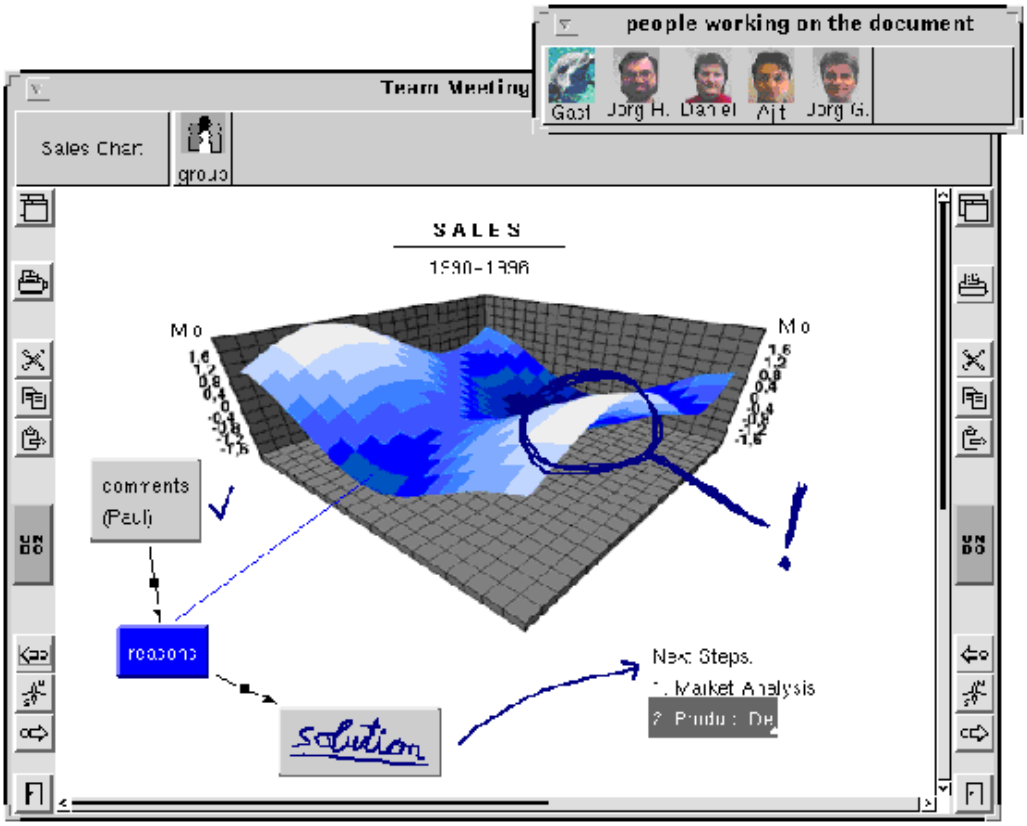

Figura 5.12: Pizarrón Compartido en DOLPHIN

En la figura 5.12 se muestra un *snapshot* del pizarrón colaborativo de DOLPHIN, el cual es compartido por todos los usuarios. Cualquier cambio que realicen sobre el mismo, se verá reflejado en las interfases de los demás participantes. No ocurre lo mismo en el caso en que los cambios se realicen en el entorno privado. Se ha especificado que los nodos antes mencionados observan a un conjunto de contenidos de documento que es lo que están mostrando en un momento determinado.

5.3.3 Modelo Colaborativo

Para plantear el modelo colaborativo de DOLPHIN y de cualquier otra aplicación, primero deben identificarse los nodos a los que se les agregará alguna funcionalidad colaborativa, ellos se representarán como nodos colaborativos. En este ejemplo, tanto el entorno compartido (pizarrón) como el espacio privado poseen características colaborativas. El nodo colaborativo que re-

presenta al pizarrón, *WhiteBoardCollaborativeNode*, observa al nodo navegacional *WhiteBoardNode* al cual agrega características tales como *awareness* de presencia e identificación de usuarios dado que muestra los distintos participantes de la reunión (ver parte superior derecha de la figura 5.12). A su vez, los cambios realizados por un usuario en el documento compartido se ven reflejados en las interfases del resto. Es por ello que también posee *awareness* de acción.

Para representar el espacio privado se ha incluido el nodo colaborativo *PrivateSpaceCollaborativeNode*, en el que un usuario puede estar manipulando un documento. Este nodo ha sido considerado colaborativo puesto que una de las características colaborativas que debe proveer es el hecho de mostrar información de *awareness*. El usuario, a pesar de estar solo editando un documento, puede también estar al tanto de cambios que están realizando otros usuarios en otros documentos.

Como se especificó en el capítulo 3, las características de *awareness* quedan a cargo del *AwarenessManager*, en este caso representado por la clase *DOLPHINAwarenessManager*, responsable de actualizar a los distintos nodos con la información de *awareness* a mostrar. Para esto, es necesario que los nodos interesados en eventos de *awareness*, se suscriban como dependientes del *DOLPHINAwarenessManager*, y además es necesario que implementen las interfases correspondientes a los eventos por los cuales quieren ser notificados, según la implementación descripta en la sección 5.1. En este ejemplo ellas son *UserNotifiableNode*, que incluye mensajes correspondientes al aviso sobre la presencia de un usuario en el nodo, y *ActionNotifiableNode*, que incluye mensajes para avisarle al nodo los cambios que se han efectuado en el documento. En la figura 5.13 se muestra cómo el nodo *WhiteBoardCollaborativeNode* implementa dichas interfases.

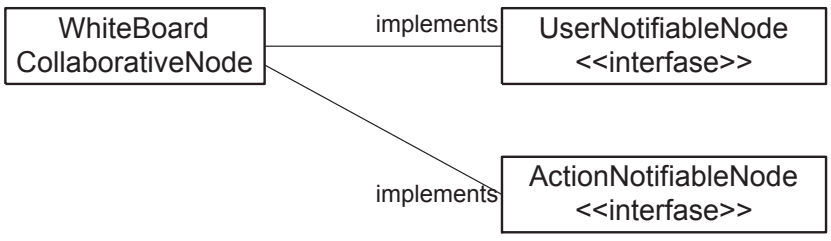

Figura 5.13: *Awareness* en DOLPHIN

Por otro lado, tanto en el nodo que representa al pizarrón compartido *WhiteBoardCollaborativeNode* como en el nodo de espacio individual representado por la clase *PrivateSpaceCollaborativeNode* es necesario utilizar una

herramienta colaborativa sincrónica: el *editor DOLPHIN*. El mismo se encuentra modelado mediante la clase *DOLPHINEditor*. También es posible que los participantes de la reunión se envíen *e-mails* entre sí. Esto fue modelado con la inclusión de otra herramienta, esta vez asincrónica, representada por la clase *DOLPHINMailTool*.

Por su parte, los nodos colaborativos ofrecen dos tipos de sesión: la correspondiente a la sesión de trabajo en grupo, *WhiteBoardSession*, que permite a un grupo de usuarios compartir la edición de un documento y que es accesible a través del nodo colaborativo *WhiteBoardCollaborativeNode*, y el tipo de sesión *PrivateSession*, que es el reservado para la edición privada de un documento, que sólo puede hacerse en el entorno privado de un participante.

5.4 Conclusiones

En las secciones anteriores se han mostrado los tres modelos propuestos por CHDM aplicados al diseño de dos aplicaciones de hipermedia colaborativas concretas.

En el primer ejemplo, Portinari colaborativo, se mostró cómo se puede agregar características colaborativas a una aplicación de hipermedia existente diseñada con OOHDM. Se vio que teniendo los modelos conceptual y navegacional previos sólo hace falta especificar qué nodos se quieren convertir en colaborativos y qué aspectos se desea agregarles.

En el caso en que no se contara con un modelo de OOHDM existente, se podría realizar un proceso de retroingeniería para construirlo a partir de la aplicación de hipermedia. Así, una vez logrados los modelos conceptual y navegacional, se puede proceder a construir el modelo colaborativo correspondiente. Tal fue el caso del segundo ejemplo, DOLPHIN, el cual consistió de una aplicación de hipermedia colaborativa ya existente, para la cual no se había realizado un diseño con OOHDM. La misma pudo ser modelada utilizando las características colaborativas planteadas en el metamodelo colaborativo de CHDM. Aquí se evidencia la ventaja de que la etapa colaborativa esté desacoplada con las anteriores y se construya de manera separada.

Las características colaborativas presentes en el primer ejemplo fueron: *awareness*, sesiones de colaboración, distintos tipos de acoplamiento en la navegación y herramientas colaborativas asociadas a los nodos. En el segundo ejemplo se incorporaron herramientas colaborativas (el editor DOLPHIN y la herramienta de *e-mail*), sesiones de trabajo y *awareness*.

Además, se mostró una posible implementación del mecanismo de *awareness* dentro de la aplicación. Dicha implementación cumple con el metamode-

lo de CHDM instanciando las metaclases correspondientes y agregando otras clases, y podría utilizarse para la implementación de cualquier aplicación de hipermedia colaborativa, dado que no depende de ninguna arquitectura ni lenguaje de programación en particular.

Capítulo 6

Comentarios Finales

En este capítulo se resumirán los resultados obtenidos en este trabajo de grado, de manera de presentar al lector un panorama general del mismo.

Primero, en la sección 6.1, se presentarán algunos trabajos relacionados con los principales temas tratados en el presente trabajo. Los mismos se han separado según tengan que ver con métodos formales y formalización de lenguajes y notaciones, o estén relacionados con propuestas de extensiones a la metodología de diseño OOHDM.

Luego, en la sección 6.2, se describirán los resultados obtenidos y conclusiones sobre ellos. Por último se enumerarán algunas líneas de trabajo futuro, posibles continuaciones a esta tesis, en la sección 6.3.

6.1 Trabajos Relacionados

En esta sección se describirán algunos trabajos relacionados con aspectos mencionados en este trabajo de grado. Sin embargo, no se han encontrado trabajos con el mismo enfoque hacia la definición de una metodología de diseño para aplicaciones de hipermedia colaborativas.

Existen diversos enfoques relacionados con el tema de formalización de notaciones. Hay mucho trabajo hecho en cuanto a encontrar una semántica precisa para UML. Como ejemplo, remitirse a [Pon99], donde se define una teoría en Lógica Dinámica para formalizar UML tanto desde el punto de vista del modelo como del metamodelo. En [KC99], se presenta una base formal para las estructuras estáticas y la semántica de los constructores básicos de los diagramas de clase UML. Las estructuras sintácticas de los constructores UML y las reglas para desarrollar un diagrama de clases bien formado son descriptas con precisión utilizando la notación Z. Sobre la base de dicha descripción formal, los constructores de UML son luego traducidos

a constructores Object-Z. Por su parte, [BKM99] propone una extensión a UML como notación para modelar aplicaciones de hipermedia diseñadas para Internet, basándose en las etapas provistas por OOHDM. Estos trabajos se relacionan con el capítulo 4, donde se dio una especificación formal para las construcciones de la metodología CHDM propuesta.

Por otro lado, cabe mencionar los trabajos relacionados a elaborar extensiones de la metodología OOHDM. En [JCBZ99] se plantea una posible extensión a OOHDM para diseñar aplicaciones colaborativas, de manera informal. También se ha utilizado OOHDM como punto de partida para diseñar aplicaciones pensadas para Internet [SREL, SR98], así como también para plantear patrones que conllevan al reuso de diseño y no sólo de implementación [RSL00].

6.2 Conclusiones

El principal objetivo de esta tesis fue definir una extensión a la metodología OOHDM de tal forma de soportar el diseño de aplicaciones de hipermedia colaborativas. Para esto primero se desarrolló un modelo (capítulo 3) para representar las distintas etapas descriptas en la metodología OOHDM (capítulo 2). Dichas etapas (o modelos) se encuentran definidas de manera informal en [Ros96], utilizando simplemente el lenguaje natural y algunos diagramas de clase. Es por eso que se vio la necesidad de clarificar los conceptos que se plantean utilizando una notación algo más formal: diagramas de UML.

Se definió una extensión a OOHDM para el soporte de diseño de aplicaciones de hipermedia colaborativas. La misma, denominada CHDM (Colaborative Hypermedia Design Method), reusa las etapas de diseño de OOHDM y propone agregar una etapa más, llamada colaborativa, en la que se tienen en cuenta las características colaborativas de la aplicación a diseñar. Primero se emplearon diagramas de clase de UML para representar los aspectos estructurales de cada una de las etapas y posteriormente se utilizaron diagramas de secuencia para modelar aspectos dinámicos de comportamiento en la nueva etapa propuesta por CHDM. Más tarde se utilizó el lenguaje de especificación formal Object-Z [Gri97, RGG94, KC99] como formalismo para detallar cuestiones más finas que no se deducían claramente de la especificación hecha en UML, y se mostró un ejemplo de las posibles verificaciones formales que pueden realizarse sobre cualquier aplicación diseñada con CHDM (sección 4.3.2).

En el capítulo 5 se mostraron los resultados obtenidos mediante ejemplos. Se utilizó la nueva metodología CHDM para construir el diseño de dos aplicaciones de hipermedia colaborativas: una aplicación de hipermedia pre-

viamente diseñada con OOHDM, el Museo de Portinari [TS], a la que se le agregaron características colaborativas, y una aplicación de hipermedia colaborativa existente, DOLPHIN [SHH94], para la cual se realizó retroingeniería para construir las etapas de diseño de la metodología propuesta, dado que no se contaba con un diseño en OOHDM previo.

En conclusión, el presente trabajo de grado provee una metodología de diseño para aplicaciones de hipermedia colaborativas, con una base formal subyacente que permite realizar verificaciones formales sobre los modelos de tales aplicaciones construidos con dicha metodología.

En las secciones siguientes se describen las contribuciones de CHDM y los beneficios de haber provisto una base formal a la misma.

6.2.1 Contribuciones de CHDM

Los aportes de CHDM para suplir las carencias de OOHDM (sección 3.2.1) para soporte de diseño de aplicaciones colaborativas son:

- Definir una jerarquía de usuarios y roles de usuarios que puede ser instanciada de acuerdo a las necesidades de la aplicación. Además permite especificar las características de *awareness* para cada nodo colaborativo.

- Permitir asociar a los nodos colaborativos herramientas colaborativas sincrónicas o asincrónicas.

- Permitir modelar sesiones de interacción y colaboración entre usuarios. Para eso se propuso la definición de tipos de sesiones accesibles a partir de los diferentes nodos colaborativos de la aplicación.

El diseñador de aplicaciones de hipermedia colaborativas sólo tiene que instanciar el metamodelo propuesto por CHDM para modelar las características colaborativas mencionadas en los puntos anteriores.

En conclusión, CHDM extiende la metodología de diseño OOHDM para aplicaciones de hipermedia colaborativas, modelando las características colaborativas de tales aplicaciones que OOHDM no tiene en cuenta.

6.2.2 Contribuciones de la Formalización

En el capítulo 3 se mostró una formalización mediante diagramas de objetos en notación UML para la metodología de diseño CHDM. Al ser UML un lenguaje de especificación gráfico semi-formal cuenta con las desventajas propias de tales lenguajes: posibles interpretaciones ambiguas, inconsistencia en

distintos modelos de un mismo sistema, etc. Por lo tanto en el capítulo 4 se mostró una especificación de CHDM mediante un lenguaje formal (Object-Z), permitiendo así que los modelos de aplicaciones de hipermedia colaborativas diseñados con la metodología CHDM estén precisamente especificados.

El hecho de haber especificado formalmente el modelo de CHDM con Object-Z da lugar a las siguientes ventajas:

- La especificación en Object-Z provee una base formal a la metodología CHDM.
- Como CHDM conforma un metamodelo para las aplicaciones de hipermedia colaborativas, dicho metamodelo puede ser instanciado a la hora de realizar un modelo para una aplicación en particular. Una vez instanciado el modelo de CHDM puede verificarse el cumplimiento de las reglas de buena formación especificadas para las metaclases. Por lo tanto, se provee una base formal para todos los diseños de aplicaciones de hipermedia colaborativas.
- Podría implementarse un editor de CHDM que permitiera verificaciones dentro de los modelos generados por éste basándose en el formalismo subyacente y los invariantes especificados.

6.3 Trabajo Futuro

El presente trabajo de grado ha dejado una metodología propuesta para el desarrollo de aplicaciones de hipermedia colaborativas. Actualmente existen diversos puntos por explotar, tanto a nivel de diseño de la metodología como a nivel de formalización del lenguaje que emplea la misma. Para esto se proponen las siguientes ramas como continuación de este trabajo:

- Análisis de la etapa de diseño de ADVs, propuesta por OOHDM en la etapa de la interfase abstracta, y el impacto de la misma en el *awareness* de la interfase.
- Implementación de un editor de CHDM que permita verificaciones dentro de los modelos generados por éste basándose en el formalismo subyacente y los invariantes especificados.
- Estudio detallado del diseño de las posibles herramientas colaborativas que pueden incluirse en los nodos colaborativos, tanto sincrónicas como asincrónicas.

- Construcción de una aplicación completa utilizando la metodología CHDM a lo largo de todas sus etapas para detectar ventajas y falencias de la misma.

Apéndice A

UML

UML es un lenguaje gráfico para especificar, construir, visualizar y documentar los componentes de un sistema de software. Está basado en los conceptos de Booch, OMT y OOSE. Conforma un lenguaje de modelización ampliamente utilizado por usuarios de estos métodos y otros. Además existen diversas herramientas para su edición aceptadas por la comunidad de diseñadores y desarrolladores de software.

UML está focalizado en estandarizar un lenguaje de modelización, y no en estandarizar los procesos de desarrollo. Cada metodología en particular utilizará un conjunto de elementos del lenguaje de UML. De hecho, el esfuerzo de unificación se ha centrado sobre un metamodelo común (unificación de semánticas) y sobre una notación común. UML estandariza conceptos básicos de modelización.

Los objetivos perseguidos para la creación de UML son los siguientes:

- Proveer a los usuarios un lenguaje de modelización expresivo y listo para usarse para que puedan desarrollar e intercambiar modelos.
- Proveer mecanismos de extensibilidad y especialización a los conceptos básicos. UML seguramente deberá ser personalizado para dominios específicos. Al mismo tiempo, no se quiere que los conceptos básicos sean redefinidos o reimplementados para cada área de aplicación.
- Mantener independencia de lenguajes de programación y procesos de desarrollo.
- Proveer conceptos de desarrollo de más alto nivel como colaboraciones, ambientes de trabajo, patrones y componentes.
- Una motivación clave detrás del desarrollo de UML ha sido la integración de las mejores prácticas de la industria, logrando tener varias vistas

basadas en niveles de abstracción, dominios, arquitecturas, etapas de desarrollo del software, técnicas de implementación, etc.

Las distintas vistas de un modelo están definidas en UML a través de los distintos tipos de diagramas que incluye. Estos diagramas proveen múltiples perspectivas de un sistema que está siendo analizado o desarrollado. Los diagramas constituyen proyecciones sobre los elementos de un modelo. Son las principales herramientas que utilizan los desarrolladores al manipular modelos.

A.1 Diagramas

Los principales diagramas incluidos en UML caen dentro de tres categorías: de Casos de Uso, de Estructura Estática y de Comportamiento. A continuación se explicará brevemente cada una de ellas, junto con ejemplos.

A.1.1 Diagramas de Casos de Uso

Los diagramas de Casos de Uso son diseñados a fin de contener la información del análisis de requerimientos. Está conformado por actores y casos de uso. Los actores representan las entidades que interactúan o intercambian información con el sistema. Los casos de uso especifican una secuencia cualquiera de transacciones llevadas a cabo cuando un usuario dialoga con el sistema.

A.1.2 Diagramas de Estructura Estática

Estos diagramas muestran la estructura estática del modelo, su estructura interna y sus relaciones. No muestran información temporal; sin embargo pueden contener referencias a estructuras que sí lo hagan.

Dentro de los diagramas de estructura estática, se encuentran los diagramas de clases y de objetos. A continuación se explican en detalle.

Diagramas de clases

Un diagrama de clases proporciona una visión gráfica de la estructura estática del modelo. Un diagrama de clases es una colección de elementos (estáticos) declarativos del modelo, tales como clases, interfases, y sus relaciones, interconectados unos con otros.

En la figura A.1 se muestra un ejemplo de un diagrama de clase de UML. Allí se muestran las clases *Company* y *Worker* unidas por la relación *Job*, la cual también es representada por una clase.

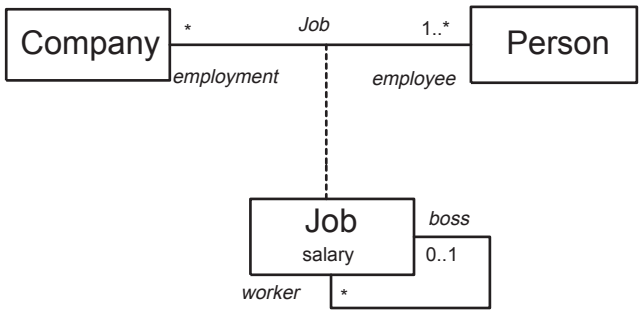

Figura A.1: Diagrama de Clases

Diagramas de objetos

Un diagrama de objetos es un gráfico de instancias, que incluye objetos y valores (datos). Un diagrama estático de objetos es una instancia de un diagrama de clases; proporciona una visión momentánea y detallada del estado del sistema en un punto determinado del tiempo.

Una clase es notada como un rectángulo con tres compartimientos separados por líneas horizontales. El compartimiento superior contiene el nombre de la clase y otras propiedades generales de la misma. El segundo compartimiento contiene una lista de atributos y finalmente el compartimiento inferior contiene una lista de operaciones.

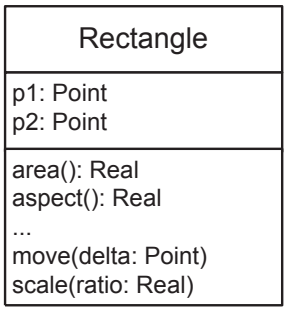

Figura A.2: Diagrama de Objetos

En la figura A.2 se muestra un ejemplo de la clase *Rectangle* representada en UML. La misma cuenta con los atributos $p1$ y $p2$ de la clase *Point*, y un conjunto de métodos tales como *Area*() o *Aspect*() para obtener el área o el aspecto del rectángulo.

A.1.3 Diagramas de Comportamiento

Los diagramas de comportamiento se centran en la dinámica del sistema. Muestran la interacción entre objetos y sus interrelaciones. El comportamiento es reflejado mostrando la interacción de un grupo de clases. Muestra relaciones que conectan clases de una interacción modelando un mensaje que es enviado entre ellas. Este mensaje, a su vez, representa una acción a llevarse a cabo. Estos diagramas incluyen dos especializaciones: de interacción (incluyendo a su vez, a los de colaboración y secuencia) y de estados.

Cada diagrama de interacción representa un escenario posible incluido en la funcionalidad total del sistema. Muestra un conjunto de interacciones entre objetos en una única ejecución de un sistema.

Diagramas de Colaboración

Un diagrama de colaboración es una construcción que se enfoca en los objetos y mensajes involucrados en cumplir un propósito. Enfatiza las relaciones entre los objetos. Se usan para describir los efectos externos provocados por una entidad a la cual está relacionado un objeto. También se usan para mostrar cómo un ítem está implementado internamente.

Diagramas de Secuencia

Un diagrama de secuencia enfatiza la temporalidad de los mensajes mostrando explícitamente su secuencia. En particular, muestra a los objetos participantes y a sus "líneas de vida" así como también los mensajes que envían y reciben entre ellos. No muestra las asociaciones entre los objetos.

Además, un diagrama de secuencia puede presentarse básicamente de dos formas: una es genérica (describe todas las posibles secuencias) y la otra es instanciada (describe una secuencia particular consistente con la forma genérica). Un diagrama de secuencia tiene dos dimensiones: la dimensión vertical representa el tiempo y la dimensión horizontal representa los diferentes objetos. El orden horizontal en el que se ubican los objetos no tiene significado alguno.

En la figura A.3 se muestra un ejemplo de este tipo de diagramas. Allí se describen varios escenarios diferentes, tales como la creación o destrucción de un objeto, las líneas de vida, los envíos de mensajes, los marcos de activación provocados por los mismos, etc.

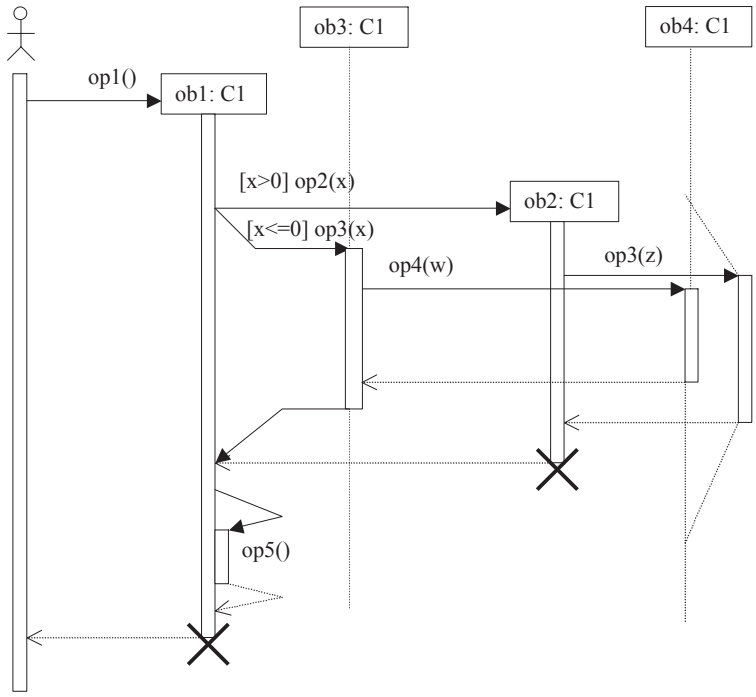

Figura A.3: Diagrama de Secuencia

Diagramas de Estados

Los diagramas de estados muestran una secuencia de estados por los que un objeto o una interacción puede pasar durante su existencia en respuesta a estímulos recibidos y también muestra sus respuestas y acciones. Estos diagramas son utilizados generalmente para representar sistemas de tiempo real u objetos con comportamientos variados que pueden pasar por diferentes estados.

Apéndice B

Object-Z

Object-Z es una extensión del lenguaje formal de especificación Z [DKRS91, Spi92]. A pesar de que Z contribuye positivamente a la especificación formal de software, adolece de algunos problemas a la hora de especificar grandes sistemas. De este modo, surge una extensión denominada Object-Z, que posee características de Orientación a Objetos, con todas las ventajas de claridad que esto supone, mejorando la legibilidad de grandes especificaciones, con posibilidad de verificación modular y refinamientos.

Z es un lenguaje de especificación de sistemas de software; permite hacer uso de la abstracción de representación, es decir, la abstracción de tipos de datos de alto nivel (conjuntos, relaciones, funciones) y de la abstracción procedural, que significa centrarse en qué debe hacerse y no cómo.

Z es un lenguaje cuya gramática y semántica están basadas en la matemática. Más precisamente, la semántica se basa en la matemática clásica, diferenciándose de otros lenguajes de especificación basados en matemáticas no clásicas. Las especificaciones en Z incluyen texto matemáticamente preciso con texto informal explicativo. El material informal facilita la interpretación del texto formal por relacionarlo al mundo real. La ventaja de la formalidad es la eliminación de la ambigüedad inherente a la descripción informal y la provisión de una base rigurosa para el razonamiento.

El componente principal de Z es el esquema. Éste se denota usualmente como una caja dividida en dos secciones. La primera contiene las declaraciones y la segunda el predicado expresado en lógica de primer orden. Una especificación define un conjunto de esquemas de estado y de operaciones. Un esquema de estado agrupa variables y define las relaciones que se deben respetar entre sus valores. En cualquier instante, estas variables definen el estado de la parte del sistema que ellas modelan. Un esquema de operación define la relación entre el estado anterior y posterior correspondientes a uno o más esquemas de estado. En Z, inferir los esquemas de operación que pue-

den afectar a un esquema de estado particular requiere examinar todos los esquemas de operación, lo cual es impracticable en especificaciones grandes. Object-Z soluciona este problema introduciendo una estructura de clase que encapsula un único esquema de estado con las operaciones que pueden afectar a ese estado. Cada clase puede ser examinada y entendida en forma separada. Las clases complejas pueden ser especificadas en términos de clases más simples estructurándolas por medio de herencia e instanciación.

B.1 Notación

Una clase se define de la siguiente manera:

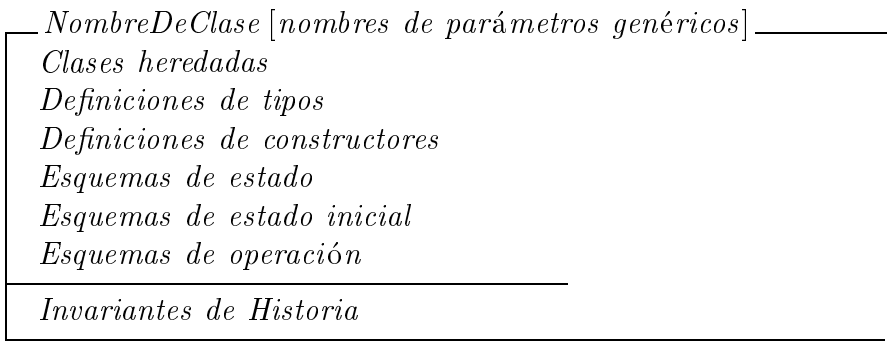

Figura B.1: Template de Diagrama de Clase

El esquema de estado no tiene nombre explícito y contiene declaraciones (las variables de estado) y un predicado (invariante de estado). Las variables de estado y constantes son llamadas atributos. Las declaraciones de tipos definen tipos locales a la clase. Los atributos y el predicado del esquema de estado están implícitamente incluidos en las secciones de declaración y predicado respectivamente del esquema de estado inicial y de cada esquema de operación. Los invariantes de estado valen en todo momento, es decir, en cada posible estado inicial y luego de cada operación. El esquema de estado inicial es distinguido por tener como nombre la palabra clave *INIT*.

Los esquemas de operación en Object-Z difieren de los esquemas de operación de Z puesto que tienen además de las secciones de declaración y predicado una lista $\triangle$ que contiene las variables de la declaración cuyos valores pueden cambiar cuando la operación es aplicada a un objeto de la clase. Los atributos y operaciones son llamados *características (features)*. El invariante

de historia es típicamente un predicado temporal que restringe aún más el comportamiento de los objetos de la clase.

Los esquemas se escriben en un formato de caja particionada en una sección de declaraciones y una de predicado expresado en lógica de primer orden, para lo cual se utilizan los símbolos tradicionales $\vee, \wedge, \Rightarrow, \Leftarrow, \forall, \exists, \neg$. Si el predicado es omitido es implícitamente verdadero. Una instancia de un esquema une los valores de las variables del esquema consistentemente con sus tipos y las restricciones.

Los esquemas de operación modelan transiciones de estado relacionando los valores de las variables antes de la operación (pre-valores) a sus valores después de la operación (post-valores). Los post-valores son primados ($valor'$) para distinguirlos de sus correspondientes pre-valores. La operación puede tener inputs (distinguidos por nombres terminados en "?") y outputs (distinguidos por la terminación "!").

Los tipos de datos en Z están basados en conjuntos, se permite la introducción de tipos arbitrarios, y se construyen nuevos utilizando constructores para conjuntos potencia "$\mathbb{P}$", productos cartesianos "$\times$" y esquemas (basados en los tipos básicos). Una secuencia en Z es formalizada como una función con dominio en los números naturales desde 1 hasta la longitud de la secuencia. El conjunto de todas las secuencias con elementos de un conjunto $\mathbb{X}$ sería:

$$\mathrm{seq}\,\mathbb{X} == \{s : \mathbb{N}_1 \rightarrowtail \mathbb{X} \mid \exists\, n : \mathbb{N} \bullet \mathrm{dom}\, s = 1..n\}$$

donde "seq" denota "secuencia de", "==" denota equivalencia sintáctica, "$\mathbb{N}_1$" representa los números naturales excluyendo al cero, "|" y "$\bullet$" se leen "tal que" y "dom" abrevia "dominio de".
Se definen las funciones *head* y *tail* con las siguientes equivalencias:

head s = s(1) y s = $\langle$ head s $\rangle \frown$ tail s

donde "$\frown$"denota concatenación de secuencias.
Existen dos formas de reuso de clases: instanciación y herencia. Utilizando la instanciación una clase puede utilizarse como un tipo del sistema de tipos de Z. De esta manera *o:Class* declara un objeto *o* de la clase *Class*. Se puede acceder a las características de un objeto (ya sean atributos o métodos) mediante la notación puntual. Es muy frecuente que los objetos tengan objetos de otras clases como variables de instancia. Utilizando herencia, una clase se utiliza como base para definir otras clases de manera incremental. Las características de la clase se incorporan a las de la nueva por medio de herencia. Las definiciones de tipos y constantes de las clases heredadas (puede existir herencia múltiple) y los declarados explícitamente en la nueva clase se fusionan. Sucede lo mismo con los esquemas, los del mismo nombre se unen,

al igual que los invariantes de historia. Sintácticamente la herencia se logra incluyendo los nombres de las clases heredadas dentro de la clase derivada. Adoptando la noción de incorporar y unir esquemas de igual nombre, heredar de más de una clase equivale a heredar progresivamente de cada una de las clases en un orden arbitrario. Los choques de nombres que llevan a mezclas inintencionales pueden evitarse mediante el renombre. Al heredar, un método de una superclase también puede ser redefinido o eliminado mediante las palabras claves *redef* y *remove*. En la figura B.2 se muestra una pila de la cual se removió el método Pop.

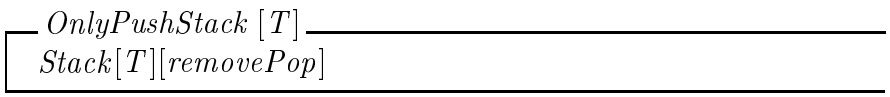

Figura B.2: Ejemplo: Only push stack

Otros operadores:

- $\Delta(x)$ se utiliza dentro de la parte de declaraciones de una operación para indicar que la variable x podría sufrir cambios durante tal operación.
- $Op_1 \bullet Op_2$ indica que Op_2 es una expresión que se evalúa en el contexto de la operación Op_1. La operación resultante es la conjunción de Op_1 con la operación correspondiente a Op_2.
- $\downarrow A$ indica el conjunto de todos los posibles objetos de la clase A y cualquiera de sus clases derivadas.
- $Op_1 \ {}^{o}_{9} \ Op_2$ indica que la salida de Op_1 se utiliza como entrada de Op_2 siempre que los parámetros coincidan en sus nombres.

Bibliografía

[BKM99] H. Baumeister, N. Koch, and L. Mandel. Towards a UML Extension for Hypermedia Design. *Lecture Notes in Computer Science. UML 99, The Unified Modeling Language*, pages 614–629, 1999.

[Boo94] G. Booch. Object-Oriented Analysis and Design with Applications, 1994.

[BRSW00] D. Bäumer, D. Riehle, W. Siberski, and M. Wulf. Role Object. *Pattern Languages of Program Design*, 2:15–32, 2000.

[CGG91] C.A.Ellis, S.J. Gibbs, and G.L.Rein. Groupware, some issues and experiences. *Communications of the ACM*, 34(1):38–58, Jan 1991.

[CMPR00] A. Cibrán, V. Mola, C. Pons, and W. Russo. Building a Bridge Between the Syntax and Semantics of UML Collaborations. *Proceedings of the ECOOP (European Conference on Object Oriented Programing) 2000*, August 2000.

[DKRS91] R. Duke, P. King, G. Rose, and G. Smith. The Object-Z specification language: Version 1. Technical report 94-45, Software Verification Research Centre, Department of Computer Science, University of Queensland, St. Lucia 4072, Australia, April 1991.

[FSP93] F.Garzotto, D. Schwabe, and D. Paolini. HDM-A Model Based Approach to Hypermedia Application Design. *ACM Transaction on Information Systems*, 11(1):1–26, Jan 1993.

[GF98] Luis A. Guerrero and David A. Fuller. Design Patterns for Collaborative Systems. In *Proceedings of the String Processing and Information Retrieval Symposium and International Workshop on Groupware*, 1998.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts, 1994.

[GMD] GMD - IPSI. German National Research Center for Information Technology. Institut Integrierte Publikations-und Informationssisteme.
`http://http://www.darmstadt.gmd.de/IPSI/`.

[Gri97] Alena Griffiths. Self-Conscious Objects in Object-Z. Technical Report 97-26, Software Verification Research Centre, School of Information Technology, The University of Queensland, Brisbane 4072, Australia, November 1997.

[Gro99] Object Management Group. The Unified Modeling Language (UML) Specification, 1999.

[HHL+92] Jörg M. Haake, Jörg Hannneman, Andreas Lemke, Wolfgang Schuler, Helge Schüt, and Manfred Thüring. SEPIA: A Cooperative Hypermedia Authoring Environment. *Proceedings of the ACM Conference on Hypertext*, 1(614920), August 1992.

[JAB96] Jr John Alfred Boyd. *Floor Control in Synchronous Groupware*. PhD thesis, The Ohio State University, 1996.

[JCBZ99] J.Schümmer, C.Shucmman, L.M. Bibbó, and J.J. Zapico. Collaborative Hypermedia Design Patterns in OOHDM. *Proceeding of the 4th International Workshop of Hypermedia Design Patterns. Darmstadt, Germany*, 1999.

[Jon90] C. Jones. *Systematic software construction using VDM*. Prentice Hall, Englewood Cliffs (NJ), USA, 1990.

[JW97] R. Johnson and B. Woolf. The Type Object Pattern, 1997.

[KC99] S. Kim and D Carrington. Formalizing the UML Class Diagrams using Object-Z. *Proceedings UML ´99 Conference, Lecture Notes in Computer Sciencie 1723*, 1999.

[MK90] G. Lausen M. Kifer. F-logic: a higher order language for reasoning about objects, inheritance and scheme. *Proceedings of the ACM SIGMOD symposium on principles of database systems*, 18(6), 1990.

[Nie95] Jakob Nielsen. *Multimedia and Hypertext, the Internet and Beyond*. Academic Press, Cambridge, 1995.

[PB00] Claudia Pons and Gabriel Baum. Formal foundations of object-oriented modeling notations. In IEEE Computer Society Press, editor, *3rd International Conference on Formal Engineering Methods,IEEE ICFEM 2000, University of York, York, UK*, 2000.

[Pon99] Claudia Pons. *Una teoría dinámica orientada a objetos como fundamento formal para el proceso de desarrollo de software basado en modelos*. PhD thesis, Facultad de Ciencias Exactas de la UNLP. Buenos Aires, Argentina, Agosto 1999.

[RG96] G. Rossi and A. Garrido. A Framework for Extending Object-Oriented Applications with Hypermedia Functionality. *The New Review of Hypermedia and Multimedia*, 2:25–42, 1996.

[RGG94] Duke R., Rose G., and Smith G. Object-Z: A Specification Language Advocated for the Description of Standards. Technical report 94-45, Software Verification Research Centre, School of Information Technology, The University of Queensland, Brisbane 4072. Australia, December 1994.

[Ros96] Gustavo Rossi. *Um metodo orientado a objeto para projeto de aplicaçoes hipermidia*. PhD thesis, PUC-Rio, Brasil, julho 1996.

[RSL00] Gustavo Rossi, Daniel Schwabe, and Fernando Lyardet. Abstraction and Reuse Mechanisms in Web Application Models. In *ER Workshops*, pages 76–88, 2000.

[Sch] Daniel Schwabe. OOHDMWeb. http://www.telemidia.puc-rio.br/oohdm/oohdmweb/.

[SHH+92] Norbert A. Streitz, Jorg M. Haake, Jorg Hannemann, Andreas C. Lemke, Wolfgang Schuler, Helge Schutt, and Manfred Thuring. SEPIA: A Cooperative Hypermedia Authoring Environment. In *European Conference on Hypertext*, pages 11–22, 1992.

[SHH94] Norbert A. Streitz, Jörg M. Haake, and Jeroen Hol. DOLPHIN: Integrated Meeting Support across Local and Remote Desktop Environments and LiveBoards. In *Proceedings of the Conference on Computer-Supported Cooperative Work, CSCW'94*, 1994.

[Shn89] B. Shneiderman. Reflections of authoring, editing and managing hypertext. In E. Barrett, editor, *The society of text*. Cambridge, Mass: MIT Press, 1989.

[SM88] S. Shlaer and S. J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice-Hall, Englewood Cliffs (NJ), USA, 1988.

[Spi92] M. Spivey. *The Z notation: a reference manual, Second edition*. Prentice Hall, Englewood Cliffs (NJ), USA, 1992.

[SR98] Daniel Schwabe and Gustavo Rossi. An Object Oriented Approach to Web-Based Applications Design. *Theory and Practice of Object Systems*, 4(4):207–225, 1998.

[SREL] D. Schwabe, G. Rossi, L. Emeraldo, and F. Lyardet. Web Design Frameworks: An approach to improve reuse in Web applications.

[TS] Luiz Tucherman and Daniel Schwabe. Projeto Portinari. http://www.portinari.org.br/.