OOHQL : Un lenguaje de consulta sobre Aplicaciones Hipermediales Orientadas a Objetos.

Trabajo de grado de la Licenciatura en Informática de

Federico Arambarri, **y**
Mauricio Sansano.

Directores:
Lic. Alicia Diaz.
Lic. Silvia Gordillo.

Facultad de Informática.
Universidad Nacional de La Plata
Noviembre, 1999.

Agradecimientos:

Agradecemos ante todo a nuestras directoras Alicia Diaz y Silvia Gordillo, quienes depositaron en nosotros toda su confianza para la realización de este trabajo y todos los que surgieron a partir de él. Les agradecemos su dedicación y las largas horas compartidas en discusiones enriquecedoras, a partir de las cuales fue posible desarrollar esta tesis.

A Pablo Zanetti, quien fue una de las personas que nos brindó todo su apoyo y conocimiento sobre el OO-Navigator framework, para poder desarrollar la implementación del lenguaje de consulta.

A nuestras familias que nos apoyaron durante toda la carrera brindándonos todo lo necesario para poder desempeñarnos correctamente durante estos seis años de carrera tan intensos.

A nuestros amigos y compañeros de la carrera, con quienes compartimos momentos inolvidables tanto fuera como dentro del ámbito académico.

A todo el LIFIA, y en especial al grupo de Objetos, por el excelente ambiente de trabajo y ayuda mutua.

OOHQL: Un lenguaje de Consulta sobre Aplicaciones Hipermediales Orientadas a Objetos.

Indice de la Tesis

Apéndices

Capítulo 1
Introducción

En la actualidad, estamos asistiendo a la construcción de una nueva generación de aplicaciones hipermediales, que combinan diferentes técnicas de navegación con comportamiento computacionalmente sofisticado, y almacenamiento de grandes volúmenes de información. Un ejemplo de este tipo de aplicaciones, son los WIS (*Web Information System*), que han tenido un gran auge de desarrollo en los últimos tiempos.

La aparición de la *World Wide Web* ha originado una nueva generación de aplicaciones hipermediales distribuidas, combinando navegación con comportamiento sofisticado, aprendizaje colaborativo a distancia, diseño asistido por computadora, y entretenimiento, que son solo un ejemplo de áreas donde esta tecnología es aplicada.

Desafortunadamente, los métodos y herramientas envueltos en la construcción y utilización de este tipo de sistemas de información, son aún escasos. Aún las aplicaciones hipermediales monousuarias, carecen de un framework unificado que soporte tanto modularidad y reuso en el nivel de diseño, como poder y entornos amigables en el nivel de usuario.

La navegación de este tipo de aplicaciones es un proceso generalmente complejo, no debido al mecanismo de navegación, sino debido al gran tamaño y estructura compleja del espacio de información. Los usuarios generalmente se pierden o suelen sentirse frustrados al buscar información sobre algún tópico específico.

Una forma de facilitar el acceso a la información, es mejorando la estructura de estas aplicaciones hipermediales, mediante la utilización de primitivas de diseño de alto nivel, tales como los Contextos Navegacionales [Schwabe96] [Rossi97], que son conjuntos de nodos relacionados semánticamente que pueden ser navegados secuencialmente.

Otra aproximación para resolver el problema de acceder a la información contenida dentro de una aplicación hipermedial, es permitir capacidades de consulta sobre estas aplicaciones. Las consultas ayudan a los usuarios en la tarea de la búsqueda de información, y también ayudan a los diseñadores de este tipo de aplicaciones a desarrollar estructuras de navegación de alto nivel. Esta tesis se basa en la implementación de un lenguaje de consultas para sistemas hipermediales orientados a objetos, denominado OOHQL (*Object-Oriented Hypermedia Query Language*).

En este capítulo esta organizado de la siguiente manera: en la sección 1.1 se presenta las motivaciones de la tesis. En la sección 1.2 se presenta la justificación del porque utilizar el paradigma orientado a objetos. En la sección 1.3 los objetivos de la tesis y en la sección 1.4 un reseña de la estructura de la presente tesis.

1.1 Motivación: Ventajas de la consulta de información contenida en aplicaciones hipermediales.

El método primario para acceder a la información en aplicaciones hipermediales es a través de la navegación. Cuando una red hipermedial no conocida es pequeña y bien estructurada, es probable que a través de la navegación se pueda recorrerla y no

perderse, pero esto puede no ser verdadero en hipertextos con un gran número de nodos y relaciones de diferentes tipos, o cuando los usuarios quieren recuperar información específica. En ambos casos, el usuario probablemente tenga problemas con la navegación, porque la persona que recorre el hipermedia puede perderse en el hiperespacio, o puede ser demasiado complejo recuperar algún ítem de información.

Para ayudar al usuario se han utilizado diversas técnicas, como por ejemplo diversas clases de índices, pero el acceso por indizado sigue siendo algo complejo de utilizar si la hipermedia es demasiado extensa. La utilización de un lenguaje de consultas sobre hipermedias, provee una nueva e importante herramienta que posibilita reducir el espacio de búsqueda del usuario a nodos de su interés. Esto se logra a través de la formulación de una consulta en la cual se especifica, mediante los predicados de consulta adecuados, que nodos se desea observar.

Un camino para resolver la complejidad de la construcción de grandes aplicaciones hipermediales es utilizar un modelo de diseño (como OOHDM, RMM, etc) que aporte conceptos para construir la estructura de la hipermedia. En nuestro caso utilizamos una metodología de diseño orientada a objetos, como es OOHDM. El lenguaje de consultas permite recuperar información contenida en la aplicación hipermedial, y sirve como herramienta en la etapa de diseño y en la instanciación de los contextos navegacionales [Diaz97].

1.2 ¿Por qué utilizar orientación a objetos.

En los últimos años se han realizado numerosos esfuerzos para llegar a la definición de un modelo que permita especificar las etapas del desarrollo de aplicaciones de esta naturaleza. La experiencia ha demostrado que el costo de mantenimiento, reuso y extensión de este tipo de sistemas es mucho mayor si se aplican sistemáticamente los principios de la ingeniería de software moderna.

Una línea cada vez más estudiada, está dirigida a la utilización de metodologías de orientación a objetos para modelar e implementar estos sistemas, debido a las posibilidades que esta tecnología brinda para la representación de información compleja a través de la combinación de las ideas de encapsulamiento y abstracción de datos, y la utilización de polimorfismo que permite construir sistemas a partir de componentes potencialmente interoperables, entre otras características.

Por lo expuesto anteriormente, resulta una de las soluciones más apropiadas la utilización del paradigma de programación orientada a objetos para el desarrollo del presente trabajo.

1.3 Objetivos de la presente tesis.

El objetivo principal de nuestro trabajo es permitir, a los usuarios de aplicaciones hipermediales orientadas a objetos, formular consultas sobre la información perteneciente a los nodos y links del hipermedia, proveyendo una implementación de un componente que interprete y ejecute consultas expresadas en OOHQL.

Como segundo objetivo, planteamos la necesidad de formalizar el proceso de ejecución de consultas sobre sistemas de hipermedia, desarrollando una especificación algebraica que permita expresar las consultas dentro de un marco formal, que nos posibilite luego realizar estudios de optimización y comparaciones en forma independiente del lenguaje de consultas a alto nivel y de la implementación del sistema.

Por último, es nuestro objetivo, lograr una implementación lo suficientemente extensible y mantenible, haciendo un uso provechoso de las ventajas del paradigma orientado a objetos.

1.3.1 Consideraciones generales.

El objetivo a la hora de definir OOHQL fue tener un lenguaje de consultas orientadas a objetos que permita extraer información estructurada de una aplicación hipermedial, conservando las capacidades de navegación dentro del resultado. Fue de primordial interés lograr una similitud muy cercana a los lenguajes de consulta existentes sobre bases de datos orientas a objetos, con el objetivo de reducir el impacto de aprendizaje sobre los usuarios del lenguaje. Lo novedoso del lenguaje es la posibilidad de aprovechar todo el estudio y desarrollo de dichos lenguajes en el contexto de sistemas hipermediales. Esto resulta en la incorporación de una característica muy deseada en las aplicaciones hipermediales, que es proveer una ayuda a los usuarios durante la navegación, y una herramienta de diseño alternativa.

1.3.2 Aportes más importantes de nuestro trabajo.

Uno de los aportes más importantes de nuestro trabajo, y que nos ha llevado a la publicación de numerosos trabajos en congresos y workshops internacionales, es la definición de una nueva especificación algebraica que permite expresar consultas sobre sistemas hipermediales orientados a objetos. Dicha álgebra está basada en la definición de las álgebras relacionales de Codd [Codd70], respetando fielmente el paradigma de orientación a objetos.

Hemos definido la gramática asociada al lenguaje OOHQL, esto nos permite poder generar un parser para interpretar el lenguaje.

Por otro lado, hemos utilizado un proceso de generación de expresiones algebraicas para representar consultas, absolutamente extensible y parametrizable, que facilita notablemente la automatización del proceso de traducción de un string a una expresión algebraica de mayor nivel de abstracción.

Por último, la implementación de este lenguaje de consulta sobre la base de un framework de desarrollo de aplicaciones hipermediales, potencia y extiende las capacidades de dicho entorno de desarrollo.

1.4 Estructura de la tesis.

La presente tesis está estructurada de la siguiente forma:

- En el Capítulo 2 se describen las principales características de los sistemas hipermediales, considerando los principales elementos de estos sistemas y su campo de aplicación.
- En el Capítulo 3 se describen las principales características de los lenguajes de consulta, realizando diversas comparaciones, y detallando los elementos involucrados en la definición de los mismos.

- En el Capítulo 4 se describe el modelo de datos utilizado como estructura del repositorio de información sobre el cual se realizarán las consultas, y se especifica en forma completa el lenguaje OOHQL.
- En el Capítulo 5 se introduce el modelo de capas y las herramientas tecnológicas utilizadas en cada una, para la construcción de aplicaciones hipermediales que soporten la realización de consultas utilizando OOHQL.
- En el Capítulo 6 se presenta la especificación formal del álgebra definida para la representación de las consultas, y se establecen los pasos a seguir durante el procesamiento de las consultas.
- En el Capítulo 7 se describen los detalles más importantes de la implementación del lenguaje, y se presentan las herramientas de edición y visualización de consultas desarrolladas.
- En el Capítulo 8 damos nuestras conclusiones y comentamos cuáles son los pasos a seguir como trabajo futuro.
- El Apéndice A contiene el ejemplo que se sigue durante todo el trabajo.
- El Apéndice B contiene la especificación de la gramática del lenguaje OOHQL.

Capítulo 2
Conceptos Generales sobre Hipermedias

2.1 Introducción.

En este capítulo, y en el siguiente, definiremos algunos conceptos fundamentales estudiados durante el desarrollo del presente trabajo. En particular, este capítulo presenta los conceptos fundamentales del paradigma de desarrollo de aplicaciones hipermediales. El estudio y entendimiento de dichos conceptos, fue de vital importancia para el desarrollo de esta tesis, fundamentalmente en lo que se refiere a integrar un lenguaje de consulta con este tipo de aplicaciones.

Este capítulo está organizado de la siguiente manera: en la sección 2.2 se definen los principales conceptos que se encuentran en una aplicación hipermedial. En la sección 2.3 se presenta una división por capas de la aplicación hipermedial. En la sección 2.4 Se presentan el problema de navegación de grandes volúmenes de información y algunas de las soluciones. En la sección 2.5 se determinan algunas de las posibles utilidades de un hipermedia.

2.2 Hipermedia: Definiciones y conceptos generales.

Dentro de la gran variedad de tipos de aplicaciones existentes en la disciplina informática, las aplicaciones hipermediales han tomado gran auge en la última década, sobre todo con el advenimiento de la Web.

Un hipertexto tiene una estructura no secuencial de acceso a la información. El usuario comienza por la página inicial de la aplicación, y luego decide por cuál de los posibles caminos seguir. De esta manera, es posible acceder a un mismo elemento de información a través de diversos caminos. En la figura 2.1 se muestra un ejemplo de este tipo de estructuras.

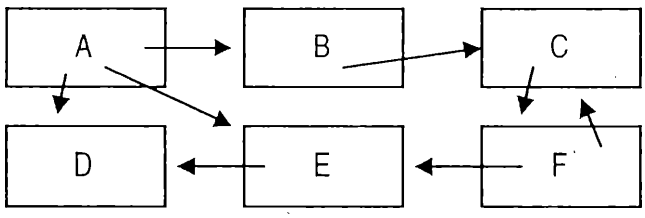


Figura 2.1. Estructura de un hipertexto.

Cada elemento de información en un hipertexto es llamado nodo. Cada nodo tiene un conjunto de punteros o enlaces que lo relacionan con otros nodos, denominados links. Todo link define un punto de partida (llamado nodo ancla) y un punto de llegada o destino (llamado nodo destino). Estos links pueden conectar un nodo con otro nodo, o

con muchos otros nodos. El conjunto de nodos vinculados mediante links, forman lo que se conoce como red del hipertexto. Los lectores se mueven a través de esta red en una actividad que a menudo suele llamarse browsing o navegación.

La definición tradicional del termino hipertexto, implica un sistema que solo muestre texto plano en sus nodos. Existen muchos sistemas actualmente, que además de texto permiten trabajar con gráficos y otro gran número de medios (ej: audio, video, etc). Es por ello que algunos autores prefieren usar el termino hipermedia para expresar la ocurrencia de aspectos multimediales dentro de una aplicación con estructura hipertextual, sin embargo, otros autores como [Nilsen94] prefieren seguir usando el termino hipertexto argumentando que este no implica la utilización sólo de texto como medio de representación de la información. En lo que a este trabajo de tesis respecta, usaremos indistintamente los términos hipermedia y hipertexto como sinónimos.

Por último, es importante mencionar que una aplicación hipermedial posee una característica denominada 'look_and_feel', la cual determina que el usuario debe sentirse, en todo momento, dueño de la información e inmerso en el sistema. Es el usuario es quien determina y controla qué información consumirá en cada instante.

2.2 Arquitectura de un sistema hipermedial.

Un sistema hipermedial puede ser dividido en tres niveles según se propone en [Campbell88]:

- Nivel de presentación: interface de usuario
- Nivel de máquina abstracta hipermedial (HAM): nodos y link
- Nivel de almacenamiento: almacenamiento de datos, manejo de datos compartidos, acceso a redes, etc. Obtención y mantenimiento de la información.

La figura 2.2 muestra una esquematización de los tres niveles mencionados anteriormente:

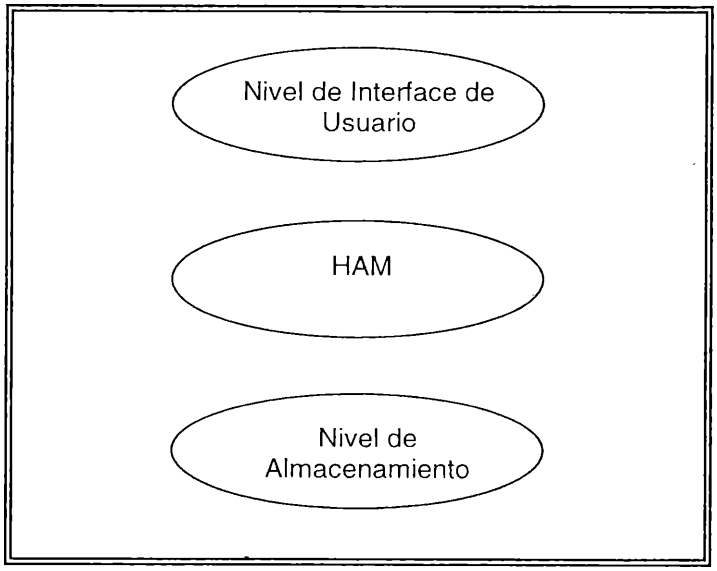


Figura 2.2: Arquitectura de un sistema hipermedial.

2.2.1 Nivel de almacenamiento

El nivel de almacenamiento, o nivel de base de datos, tiene como principal responsabilidad dar persistencia a la aplicación hipermedial. En este nivel se resuelven los problemas derivados del almacenamiento de grandes volúmenes de información, que luego serán utilizados por la aplicación hipermedial. Dicha aplicación debe funcionar en forma absolutamente independiente de la forma en la cual se provea la persistencia en este nivel.

2.2.2 Nivel de maquina abstracta hipermedial (HAM, *Hypermedia Abstract Machine*)

En este nivel se determina la funcionalidad básica de la aplicación hipermedial. Aquí se especifica cuales son los nodos y links que componen la aplicación, y cuales son las relaciones entre ellos.

El HAM es el mejor candidato para la estandarización de la importación y exportación de formatos hipertextuales, ya que en el nivel de almacenamiento la estructura depende del motor de persistencia, y en el nivel de interface pueden encontrarse varias visualizaciones por nodo.

2.2.3 Nivel de interface de usuario

La interface de usuario trata con la presentación de la información incluida en el HAM, considerando los diversos temas relevantes en la construcción de interfaces, tales como:

- comandos que deberían encontrase disponibles para el usuario,
- como mostrar nodos y links,
- si mostrar un diagrama con una visión global del sistema o no, y demás.

En el nivel de la interface de usuario se determina si se muestra o no toda la información de cada nodo al usuario.

2.4 La navegación: grandes volúmenes de información.

Como definimos anteriormente, la forma en que los usuarios pueden acceder a la información contenida en una aplicación hipermedial, es a través de la navegación. Un usuario navega a través de los nodos del hipermedia siguiendo los links que conectan un nodo con otro.

Cuando la red hipermedial está constituida por un gran número de nodos, el espacio de navegación crece considerablemente. Esto se debe a que la aplicación hipermedial almacena grandes volúmenes de información, lo cual hace que la cantidad de nodos que la componen, y sus links asociados, aumenten notablemente. Este fenómeno tiene un efecto no deseado, ya que la probabilidad de que los usuarios se pierdan durante la navegación en la red hipermedial es mayor. Esta probabilidad aumenta en forma directamente proporcional al crecimiento del espacio de navegación. El hecho de perderse durante la navegación hace que los usuarios se sientan

rápidamente desorientados y desanimados, con el consiguiente abandono de la aplicación.

Con la evolución de la teoría y las técnicas de diseño en el campo de las aplicaciones hipermediales, diversas soluciones han sido propuestas en pos de la solución al problema de la desorientación en el espacio navegacional. A continuación detallamos brevemente las más relevantes:

- **Tour Guiado:** un tour guiado está compuesto por una serie de nodos semánticamente asociados, que conducen al usuario a través de cierta parte de la aplicación. Los tours guiados pueden ser usados para introducir a los nuevos lectores a los conceptos generales del hipermedia, y también proveer diferentes tours guiados para satisfacer diferentes expectativas según el lector.

- **Backtrack:** el bactrack permite navegar hacia el nodo previamente visitado por el usuario. Casi todos los sistemas hipermediales proveen alguna forma de backtrack, pero no siempre es consistente.

  El backtrack es conceptualmente muy simple: el usuario realiza un click sobre un botón y retorna al nodo anterior. El problema surge cuando el usuario realiza backtrack más de una vez y cuando ha visitado ciertos nodos más de una vez.

  El backtrack cronológico no es eficiente, ya que se gasta demasiado tiempo revisando el mismo nodo varias veces.

- **Historia:** Algunos sistemas hipermediales tienen mecanismos de historia más generales que el simple backtrack. Contienen una lista ordenada de todos los nodos que fueron visitados. Esta lista puede ser consultada en cualquier momento para navegar a algún nodo de los que fueron previamente visitados por el usuario.

  Hay hipermedias que proveen mecanismos especiales para mostrar la historia, ya que en ella pueden encontrarse grandes cantidades de nodos y puede tornarse inmanejable si se muestra solo un listado.

- **BookMarks:** Algunas aplicaciones permiten definir un bookmark (o libro de marcas) en el cual van ubicando aquellos nodos que el usuario posiblemente quiera visitar. La diferencia entre el bookmark y la lista de historia, está dada en que los nodos en el bookmark son incorporados por el usuario, mientras que en la historia son incorporados automáticamente por la aplicación. Estos nodos, al ser elegidos por el usuario, son los considerados útiles por él, conformando una lista más pequeña y por ende más fácil de manipular.

2.4.1 Diseño Navegacional.

Si bien las herramientas descriptas anteriormente han hecho su aporte al problema de desorientación previamente descripto, la realidad indica que no han sido suficientes para solucionar dicho problema definitivamente.

Debido a esto, muchos autores han desarrollado herramientas de diseño de aplicaciones hipermediales, tales como HDM (Hypermedia Design Methodology) [Garzotto93], OOHDM (Object Oriented Hypermedia Design Methodology) [Schwabe95] y RMM (Relationship Management Methodology) [Isakowitz95].

En líneas generales, todas estas metodologías persiguen como objetivo, agrupar información semánticamente relacionada, constituyendo una red hipermedial compuesta por diversas agrupaciones de nodos. Dichas agrupaciones están compuestas por nodos relacionados fuertemente con otros nodos de la misma agrupación, pero débilmente con nodos pertenecientes a otras agrupaciones.

Para ejemplificar lo anteriormente descripto, consideremos la Figura 2.3, en la cual es fácil percibir que el esquema de la derecha es mucho más fácil de navegar que el de la izquierda.

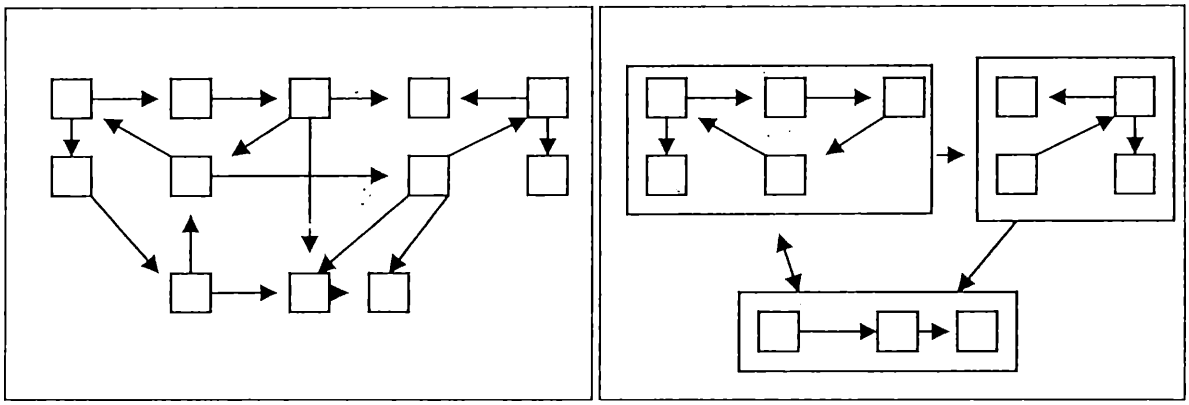


Figura 2.3: Estructuración de Hipermedias.

2.4.2 Contextos Navegacionales

Una herramienta de diseño muy potente, son los contextos navegacionales. Un contexto navegacional se pueden definir como una herramienta de alto nivel que se utiliza para ayudar al usuario en el proceso de navegación, compuesto por una colección de nodos semánticamente relacionados.

En [Schwabe98] los contextos navegacionales son presentados como elementos que permiten acceso a la información de un hiperdocumento. Los contextos minimizan los problemas de desorientación en aplicaciones hipermediales complejas. Un contexto navegacional esta conformado por un conjunto de nodos, e incluye una forma predefinida para accederlo y recorrerlo.

Se pueden navegar los mismos nodos desde diferentes contextos navegacionales, a su vez cada nodo puede tener información adicional relacionada con el significado del contexto en el cual se lo está visitando. Esto implica por ejemplo, que el nodo siguiente y el nodo anterior de un nodo, no son siempre los mismos, sino que dependen del contexto de navegación en el cual se encuentre el usuario de la aplicación hipermedial y mediante el cual fue activado el nodo.

Los contextos navegacionales se usan como un mecanismo estructurado para generar navegación durante el proceso de desarrollo. OOHDM define la aplicación hipermedial en términos de contextos navegacionales, y esto es lo que diferencia significativamente a esta metodología de las demás existentes. Es por esto, que los contextos navegacionales son considerados como primitivas de alto nivel, en el diseño de aplicaciones hipermediales.

Un contexto navegacional puede ser definido en modo extensivo, indicando cada uno de los nodos que pertenecen al contexto, o por comprensión, formado por los nodos de la aplicación hipermedial que satisfacen un cierto predicado.

En OOHDM hay varias estrategias para especificar un contexto:

- **Arbitraria:** se seleccionan arbitrariamente los nodos de la aplicación hipermedial que conformarán el contexto.
- **Derivado de una clase:** conjunto de todas las instancias que pertenecen a la misma clase de nodo. Esto puede ser especializado para que sean solo las instancias de la clase que cumplen con cierto predicado.
- **Derivado de un link:** conjunto de todas las instancias que son destino de un link 1 a n. Se genera un contexto navegacional con los n elementos del destino.(Figura 2.4)

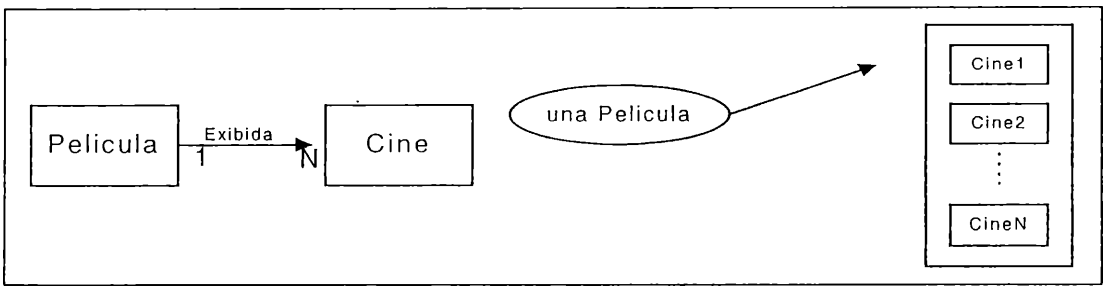


Figura 2.4: Contexto derivado de un link 1 a N.

- **Derivados de la composición:** está formado por un atributo multivaluado de un nodo, por ejemplo, una ruta compuesta por un conjunto de imágenes que la muestran (Figura 2.5)

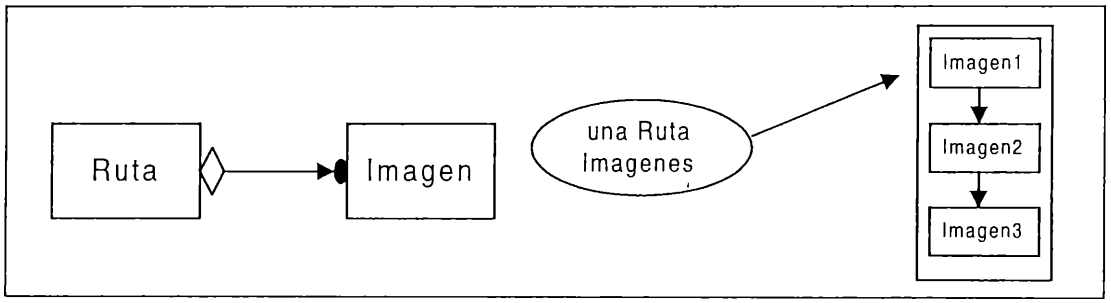


Figura 2.5: Contexto derivado de un atributo multivaluado.

Asociado al conjunto de nodos que componen un contexto navegacional, debe especificarse también la forma de acceso a dichos nodos. Los mecanismos de acceso más frecuentes son:

- Secuencial.
- Secuencial indexado.
- Random.

2.5 Aplicaciones de un Hipermedia.

En la actualidad, existe un gran interés en proveer funcionalidad hipermedial a aplicaciones convencionales. Esto se debe, entre otros factores, a la facilidad y naturalidad que las aplicaciones con características hipermediales brindan en cuanto al acceso a la información.

No todas las aplicaciones no hipermediales deberían ser transformadas en un hipermedia. Para determinar si una aplicación puede o no ser un hipermedia, en [Shneiderman89] se han propuesto una serie de reglas, conocidas como reglas de oro de un hipermedia:

- Una gran unidad de información es organizada dentro de numerosos fragmentos.
- Los fragmentos están relacionados entre sí.
- Los usuarios solo necesitan una fracción de la información en cada momento.

Veamos ahora los campos más frecuentes de utilización de las aplicaciones hipermediales:

2.5.1 Documentación en línea.

La documentación en línea sea quizás uno de las más naturales aplicaciones hipermediales, este fue el propósito de las primeras aplicaciones reales de hipermedias.

Es probable que un usuario no lea el manual de su software completamente, en general es un libro de consulta, la información que consume es puntual a la actividad que desea realizar y sus tópicos relacionados.

Es probable que el libro lo halla prestado, o no lo encuentre, o este lejos del lugar donde se encuentra trabajando. Por esto la documentación on-line tiene innumerables ventajas. Una de las mejores formas de presentar esta documentación es a través de un hipermedia, por que la cantidad de información que se consume en cada momento es pequeña y se encuentra relacionada entre sí.

2.5.2 Asistencia al Usuario.

Un hipermedia provee un mecanismo integral para proveer diferentes formas de asistencia al usuario, incluyendo manuales on-line, un tutorial introductorio, un sistema de ayudas on-line, y manejo de errores.

En un asistente de usuario basado en un hipermedia, debe ser posible para el usuario tener un link desde el mensaje de error a la localización en el sistema de ayuda que dará asistencia para resolver el problema.

2.5.3 Ingeniería de software.

Durante el ciclo de vida del desarrollo de software, se produce un largo número de especificaciones y documentos de diseño. Los hipertextos tienen un gran potencial para proveer links entre ellos.

Uno podría querer comenzar por los documentos de análisis de requerimientos y navegar hacia los documentos de diseño que representan dichos requerimientos. O desde una especificación hacia la sección de código que la implementa, inclusive podría realizarse una navegación inversa, desde una sección de código hacia el requerimiento que le dio origen.

2.5.4 Aplicaciones comerciales.

Muchas son las aplicaciones comerciales a las que se les puede dar formato hipermedial, como por ejemplo:

- Proveer asistencia a clientes, por ejemplo una fábrica de autos puede ofrecer un hipermedia que muestre el funcionamiento del automotor, junto a posibles problemas y soluciones.
- Diccionarios y libros de referencia en forma hipermedial para consulta rápida de información específica.
- Auditorías, la tarea de auditoría produce una gran cantidad de documentos y luego se analiza la información en conjunto para ver los puntos de relación y la consistencia de la información.
- Para representar conjuntos de leyes, en general estas leyes se encuentran relacionadas con otras leyes con las cuales se complementan para determinar una completa jurisprudencia. También para presentar expedientes, ya que a menudo se hace referencia a diferentes leyes aplicadas.

2.5.5 Aplicaciones intelectuales.

Entre las que se destacan:

- Hipermedias de investigación, serie de artículos científicos relacionados.
- Documentos generados por una organización.
- Periodismo, para representación de hechos con sus artículos relacionados.

2.5.6 Aplicaciones Educacionales.

Entre las que se encuentran principalmente:

- Enciclopedias.
- Museos.
- Bibliotecas.

2.5.7 Entretenimiento.

En este rubro se encuentran varias clases de juegos, y galerías de música, entre otras aplicaciones.

Capítulo 3
Lenguajes de Consulta

3.1 Introducción.

En este capítulo definiremos los conceptos generales correspondientes a los lenguajes de consulta, estudiados durante el desarrollo del presente trabajo de tesis. El estudio de dichos conceptos nos permitió desarrollar la implementación del lenguaje de consultas OOHQL en forma consistente, es decir, independizando el diseño, del lenguaje de implementación particular elegido para este desarrollo, y permitiendo reutilizar dicho diseño en cualquier otra plataforma de desarrollo orientada a objetos.

Este capítulo esta organizado de la siguiente manera: en la sección 3.2 se definen algunos conceptos generales sobre los lenguajes de consulta, se describen características de diferentes lenguajes declarativos y se las compara, haciendo hincapié en su relación con los lenguajes de consulta declarativos para hipermedias. En la sección 3.3 se determinan las posibles formas de procesamiento de una consulta. En la sección 3.4 se presentan los lenguajes sobre aplicaciones hipermediales existentes.

3.2 Lenguajes de consulta. Conceptos generales.

Para definir que es un lenguaje de consulta, definamos previamente la noción de *repositorio de información*. Como su nombre lo indica, un repositorio de información (o simplemente repositorio) es un contenedor de información organizado de alguna forma particular.

Un lenguaje de consulta permite recuperar información a partir de un repositorio, especificando un predicado de consulta a satisfacer. Si bien el estudio teórico de lenguajes de consulta es un tema ampliamente investigado, los lenguajes de consultas comenzaron a tener vital relevancia con la aparición de SQL (*Structured Query Language [C+76] [573991]*) en las bases de datos relacionales, a punto tal que hoy en día todo motor de base de datos relacional implementa algún dialecto de SQL.

Un lenguaje de consulta debe ser declarativo, lo cual significa que el usuario se concentra en **qué** información recuperar y no **cómo** recuperarla. La implementación de este tipo de lenguajes contempla un mecanismo de optimización que permite al usuario despreocuparse de la eficiencia durante la escritura de la consulta y concentrarse solo en la información a recuperar. Antiguamente, los lenguajes de consulta de las bases de datos jerárquicas y en red eran procedurales (se especifica **qué** información recuperar y **cómo** recuperarla), resultando bastante complejos de utilizar y dificultando la tarea de optimización de la consulta. El alto nivel de los lenguajes declarativos, y la división del **qué** y el **cómo** recuperar información, es lo que permite desarrollar mecanismos de optimización. El usuario se encarga del **qué** y el sistema del **cómo**.

En este capítulo presentaremos las características más relevantes de los lenguajes de consulta declarativos más representativos en la actualidad, tales como SQL y OQL (*Object Query Language*), en bases de datos relacionales y orientadas a objetos respectivamente.

3.2.1 Características de los lenguajes declarativos.

En esta sección determinaremos cuales son las características que se encuentran presentes en los lenguajes de consulta sobre bases de datos relacionales y sobre bases de datos orientadas a objetos, determinando como se encuentran estas características en un lenguaje de consulta sobre sistemas hipermediales.

- **Modelo de datos:** Quizás sea este el punto más importante a partir del cual se diferencian los lenguajes de consulta, ya que el modelo de datos determina muchas de las características que va a tener el lenguaje. No es lo mismo escribir consultas que busquen objetos relacionados por conocimiento o composición, en una base de datos orientada a objetos, a escribir consultas que busquen atributos de tuplas relacionadas en una base de datos relacional.

  El modelo relacional [Codd70] es el más simple de todos. Está compuesto por un conjunto de relaciones, a su vez cada relación están compuesta por un conjunto de atributos, y cada atributo toma valor dentro de un conjunto de valores (dominio del atributo). Esta simplicidad en el modelo, arroja como resultado un lenguaje de consultas simple. Este lenguaje toma relaciones como entrada de la consulta y retorna una relación como salida de la misma. Esto brinda la posibilidad de definir fácilmente consultas anidadas, es decir, consultas que toman como argumento otras consultas. La gran simplicidad de este modelo es una ventaja por su fácil comprensión y uso, pero también una desventaja ya que hace que el modelo sea poco expresivo. El modelo atrapa poca semántica de los datos, los trata como colecciones de atributos y cualquier combinación de atributos es válida para formar una relación, aunque no tenga ningún sentido semánticamente. La forma de vincular tuplas pertenecientes a distintas relaciones es a través de atributos, esta manera de asociar información no es del todo clara en el momento de intentar obtener tuplas relacionadas, ya que se deberían revisar todos los datos de una columna dentro de una tabla, para detectar cuales son los datos relacionados con otra tupla.

  El modelo orientado a objetos [Goldberg83] es más expresivo que el modelo de datos relacional y también más complejo. La tecnología orientada a objetos introduce el concepto de OID (Identificador Único de Objeto). En BDOO se maneja directamente el OID y se lo puede ver como una variable más que maneja el sistema. En lenguajes orientados a objetos, el OID está dado por el puntero a la región de memoria donde se encuentran alocados los objetos, usando el principio de que en una región de memoria puede haber alocado un solo objeto a la vez. Cuando se utiliza un manejador de bases de datos, en cambio, este concepto es más complejo ya que las referencias a los objetos deben referirse a ubicaciones en memoria secundaria. Además se incorporan los conceptos de polimorfismo, herencia, encapsulamiento de datos, mensajes y métodos. Estas herramientas proveen un mayor poder expresivo para modelar el mundo real, lo cual hace que el modelo de datos sea mucho más complejo y potente.

  Ahora, el lenguaje de consultas toma información de un conjunto de objetos, las expresiones de selección de datos son mucho más complejas ya que pueden incluir encadenamiento de mensajes, lo cual hará más compleja la tarea del optimizador. El principal inconveniente es cuando el lenguaje no es cerrado con respecto al modelo de objetos: en OQL se pueden seleccionar atributos de diferentes objetos como resultado de una consulta, este resultado no es un objeto ya que no pertenece a ninguna clase, es decir, podemos generar consultas cuyos resultados estén fuera del modelo de objetos de nuestra aplicación. Esto dificulta la definición

de nuevas consultas sobre el resultado de una consulta previa. La asociación entre objetos se establece por la relación de "conocimiento", que se establece a partir de referencias a los OID; desde este punto de vista es más simple encontrar los datos relacionados a un objeto en particular en el modelo orientado a objetos, en comparación con el modelo relacional.

Las aplicaciones hipermediales diseñadas con objetos, permiten utilizar la potencia de este paradigma en el diseño e implementación de la aplicación hipermedial. Una hipermedia puede ser vista como un conjunto de objetos nodo y objetos link.

Una consulta sobre este tipo de aplicaciones toma como entrada una aplicación hipermedial (compuesta por nodos y links), y selecciona un conjunto de nodos que satisfacen el predicado de consulta. Cada nodo contiene a su vez los links que parten de él. Lo importante es que el resultado es una colección de nodos y links, es decir, un hipermedia, en particular un hipermedia incluido en el original. Esto hace posible una definición natural de consultas anidadas, de manera análoga a lo que ocurre en el modelo relacional.

Otro punto interesante es tener la potencia de los modelos orientados a objetos para la representación del mundo real, esto explica el porqué crear un modelo orientado a objetos para el diseño de hipermedias. Luego el lenguaje de consulta se define sobre el modelo de hipermedias orientado a objetos.

Considerando la complejidad en la definición de un lenguaje de consulta sobre un modelo de datos en particular, cabe destacar que el modelo de hipermedias orientado a objetos permite definir un lenguaje de consulta más complejo que su par relacional, pero más sencillo que su equivalente en el modelo orientado a objetos puro. Esto se debe a que las aplicaciones hipermediales orientadas a objetos tienen una estructura más compleja que una aplicación procedural/relacional, pero más sencilla que una aplicación orientada a objetos, ya que solo constan de nodos y links.

Por último, las relaciones entre nodos en un hipermedia están dadas de dos formas distintas, la primera la constituyen las relaciones (heredadas del modelo orientado a objetos) de conocimiento y parte_de que se establecen a partir de variables de instancia. La segunda forma de relacionar nodos esta dada por la relación de navegación a través de links, esta relación de nodos es inherente a las aplicaciones hipermediales y por lo tanto surge únicamente en este contexto. Esta característica nueva del dominio debe ser absorbida de manera consistente por el lenguaje de consulta, permitiendo expresar estas relaciones en las consultas.

Además de las diferentes formas de estructurar los datos subyacentes, las relaciones que existen y la manera de acceder a la información, existe también una diferencia importante en los tipos de datos que cada uno de los modelos manipula. Los lenguajes de consultas relacionales manipulan un solo tipo de dato, la relación. La relación puede tener varias configuraciones, es decir, es distinta una relación que represente personas de una que represente autos. A pesar de esto, ambas son instancia de un único tipo de dato simple, la relación. Las consultas solo manejan relaciones como parámetros de entrada y salida. Los lenguajes de consultas de Bases de Datos Orientadas a Objetos (BDOO) pueden involucrar muchos tipos de datos diferentes, como por ejemplo, conjuntos, listas, tuplas y arreglos, entre otros. Una de las dificultades derivadas de esta diversidad de tipos de datos, es diseñar un álgebra que permita expresar las consultas sobre una BDOO. Se pueden encontrar detalles de estos problemas en [Yu98]. Pero ante esta dificultad, aún no se ha

llegado a consensuar un conjunto de operadores algebraicos que permitan expresar dichas consultas. Una aproximación muy utilizada para tratar de resolver este problema, es determinar un conjunto de operadores algebraicos para cada tipo de datos provisto por la BDOO, es decir, un conjunto de operadores para arreglos, otro para conjuntos, etc. Como resultado de esto, las álgebras orientadas a objetos y sus reglas de transformación son mucho más complejas que sus pares relacionales.

Como definimos anteriormente, el resultado de una consulta sobre una aplicación hipermedial, es una selección de nodos contenidos en el hipermedia original, con los links que parten de ellos. Puede ocurrir, que como resultado de la consulta, se obtenga una partición del conjunto de nodos del hipermedia original, donde entre las partes no hay conexión a través de links. Puede que tampoco se sepa a priori cual es el nodo inicial del hipermedia resultante luego de una consulta. Por lo tanto, hay que proveer alguna herramienta que permita navegar el resultado de una consulta con algún criterio, además de la relación de link impuesta por el diseñador del hipermedia. Aquí surge la necesidad de que aparezcan las estrategias de navegación que deben asociarse al resultado de una consulta obligatoriamente, las cuales nos permitirán ver el resultado en forma hipermedial.

- **Complejidad de los tipos de datos:** Para poder hacer referencia a un elemento del mundo real en particular, por ejemplo el auto azul patente ASD123, se utilizan distintas herramientas según el modelo de datos. En el modelo relacional se utiliza una clave primaria para identificar unívocamente esa tupla dentro de una relación (en particular la relación *Auto*), esta clave primaria es un atributo de la propia tupla y se maneja de igual manera que cualquier otro atributo. Esto es sumamente simple y no agrega ninguna complejidad a la hora de procesar la consulta.

Los objetos son entidades complejas, su identificación está dada por un OID que se transforma en un elemento fundamental en sistemas de BDOO. Muchas álgebras propuestas agregan operadores especiales para manejar el OID [Yu98] [Abiteboul95]. Como mencionamos antes, el OID es manejado por el sistema. En hipermedias orientados a objetos, los nodos y los links son objetos, por lo que también existe el concepto de OID.

- **Jerarquía de clases:** Las jerarquías de clases también constituyen un elemento complejo a la hora de procesar consultas en BDOO. Dicha complejidad no se encuentra presente en el modelo relacional ya que este no contempla jerarquías en su definición. Si bien el modelo de Entidad–Relación permite crear jerarquías de especialización y generalización, estas desaparecen (son transformadas en relaciones) durante el proceso de conversión de un modelo de Entidad–Relación a un esquema relacional.

En una BDOO, se puede requerir que una consulta retorne todas las instancias de una clase indicada, más las instancias de todas las clases que heredan de ella. Esto trae importantes problemas a la hora de definir los índices para diferentes requerimientos de acceso.

En hipermedias orientadas a objetos se mantiene el problema de las jerarquías de herencia, ya que éstas son inherentes al paradigma orientado a objetos, pero en este caso está limitado a la jerarquía de nodos.

- **Polimorfismo:** Uno de los problemas que aparecen a la hora de ejecutar una consulta eficiente en BDOO es el polimorfismo. Esta es una vital y potente

característica del paradigma orientado a objetos, que permite que mensajes diferentes con la misma semántica tengan el mismo nombre pero distinta implementación. Es decir, podemos tener diversas clases (no necesariamente dentro de la misma jerarquía de herencia) que implementen el mismo mensaje de diferentes formas, el selector del mensaje será el mismo pero el método cambiará en las diversas clases. De esta forma, solo en tiempo de ejecución se sabrá cual será el código que se va a ejecutar ante el envío de un mensaje, con lo cual no se pueden hacer estimaciones y optimizaciones en compilación. Ciertos tipos de optimizaciones que se realizan en bases de datos relacionales, en tiempo de compilación, son imposibles de realizar en una BDOO. Las bases de datos orientadas a objetos están obligadas a agregar un optimizador en tiempo de ejecución. En hipermedias orientadas a objetos la situación es análoga, ya que también utilizan mensajes polimórficos.

- **Métodos:** Otra característica del paradigma orientado a objetos es la aparición de métodos en el lenguaje de consulta como una manera más de acceder a la información. Esta característica complica el trabajo del optimizador. Los métodos proveen el principio de encapsulamiento, indispensable para el paradigma ya que trae innumerables ventajas para el posterior mantenimiento del software orientado a objetos. El problema es que el encapsulamiento dificulta la estimación del costo de ejecución de los métodos. Por este motivo algunos de los motores de bases de datos acceden a la representación interna de los objetos al realizar consultas. Esto supone en realidad, una violación del encapsulamiento del objeto en cuestión.

  Esta diferencia en el acceso a la información de los objetos, trae aparejada una discusión teórica que solo plantearemos en este trabajo, pero que no desarrollaremos ya que no es el objetivo de esta tesis.

  Quienes están a favor de acceder directamente a los atributos de un objeto, sostienen que la principal ventaja que se consigue, es que esto permite realizar optimizaciones antes de ejecutar la consulta.

  Por otro lado, quienes prefieren utilizar mensajes como único mecanismo de acceso a los atributos de un objeto, se basan en las siguientes justificaciones:

  1) permite proveer seguridad de acceso, ya que solo pueden accederse (consultarse) aquellos atributos para los que el programador de la clase halla definido mensajes que los retornen;
  2) se respeta el encapsulamiento, con la consiguiente ganancia en la etapa de mantenimiento;
  3) se favorece la consulta por tipo de objeto (ver sección 4.3.1).

3.2.2 Lenguajes declarativos: su relación con los repositorios de información.

Los lenguajes declarativos actúan sobre un repositorio de información con características propias, a partir del cual se recupera la información deseada.

Un lenguaje de consultas está fuertemente relacionado con el repositorio de información, su sintaxis y su semántica dependen del modelo de datos asociado al repositorio.

En nuestro trabajo, hemos desarrollado un lenguaje de consulta orientado a objetos. Podemos pensar un hipermedia como un gran conjunto de información con una estructura particular de datos. Los datos en un hipermedia pueden obtenerse desde una base de datos relacional, desde una base de datos orientada a objetos, desde un sistema

de archivos planos, o simplemente residir en memoria. La forma en la cual se da persistencia a la aplicación hipermedial es responsabilidad de la persona que construye dicha aplicación. Desde el punto de vista de un usuario puede verse a la hipermedia como un conjunto de nodos relacionados por links.

De esta forma, vamos a asumir, de aquí en más, que nuestro repositorio de información no es otro que la aplicación hipermedial compuesta por nodos y links.

3.3 Procesamiento de Consultas.

A pesar de las diferencias expuestas arriba entre los lenguajes de consultas basados en diferentes modelos de datos, muchas de las metodologías utilizadas para el procesamiento de consultas en el paradigma relacional, pueden ser adoptadas para el procesamiento de consultas en bases de datos orientadas a objetos y en hipermedias orientadas a objetos. Hay básicamente dos metodologías de procesamiento de consultas, la basada en operadores algebraicos y la basada en estimaciones de costo [Yu98].

La optimización basada en álgebra consta de dos partes.

- En la primera etapa la consulta entrante es transformada a una expresión algebraica, luego ésta es transformada a una expresión semánticamente equivalente pero más eficiente (esto se hace usando reglas de transformación algebraicas).
- En la segunda etapa se estudian las características físicas, tales como la existencia de índices para un acceso más rápido y demás consideraciones. Estas características son tomadas en cuenta para construir un plan eficiente de ejecución para las expresiones algebraicas generadas en la primer etapa.

La ventaja de este modelo es la *extensibilidad* y la relativa facilidad de su implementación. La desventaja está dada porque la segunda etapa está condicionada por las expresiones algebraicas generadas en la primer etapa y existe una gran posibilidad de que el plan de ejecución encontrado no sea el óptimo.

La segunda metodología, está basada en las estimaciones de costos de la recuperación y a partir de allí determinar cual es la mejor forma de ejecutar la consulta. Suele ser más efectiva que la aproximación algebraica y por ello la más usada en las bases de datos, tanto relacionales como orientadas a objetos.

3.4. Lenguajes de Consultas sobre Hipermedias

Existen diversos trabajos previos, cuyo objetivo es la construcción de un modelo para el diseño de hipermedias. Algunos ejemplos son HDM *(Hypermedia Design Model)* [Garzotto93], RMM [Isakowitz95], y OOHDM *(Object-Oriented Hypermedia Design Model)* [Schwabe95]. Cada uno de estos trabajos describe una metodología para diseño de aplicaciones hipermediales, pero ninguno define como realizar consultas sobre estas aplicaciones.

Por otro lado, existen algunos trabajos que se concentran en proveer la capacidad de consultar información estructural de un sistema hipermedial, es decir, su estructura constitutiva o topográfica, pero no la información semántica contenida en ella [Amann92], [Amann94] y [Concens90]. Algunas de estas propuestas utilizan sistemas de bases de datos relacionales como mecanismo para proveer conocimiento acerca de la

estructura del grafo y tipos de nodos. Por ejemplo [Parunak91] combina un motor de hipermedia con bases de datos relacionales, y presenta la información vía atributos atómicos, los cuales pueden ser consultados por los mecanismos provistos por el RDBMS. [Marmann92] propone el uso de una base de datos orientada a objetos como soporte para una aplicación hipermedial. En este caso el acceso a la información se logra usando el lenguaje de consultas provisto por dicha base de datos.

La WWW puede verse como una gran hipermedia donde los nodos son todos del mismo tipo, documentos hipermediales. Se están estudiando lenguajes para realizar consultas sobre este gran hipertexto. En [Mihaila96] se define un lenguaje que permite ubicar aquellas páginas que contengan algún texto en particular o la distancia (medida en links) a alguna página que contenga dicho texto. El lenguaje de alto nivel en este caso permite extraer información desde la Web tomando las múltiples ventajas de los múltiples servidores de índices existentes en la red, en forma totalmente transparente al usuario. La idea utilizada es darle a la Web una visión de base de datos relacional donde hay dos grandes tablas de documentos y de links (Figura 3.1) y a partir de allí realizar las consultas.

Document

URL	Title	Text	Length	Type	Modif
http://www...	Title1	Text1	1236	Lest	1-1-96
...	...	...	...	...	...

Anchor

Base	Label	Href
http://www...	Label	http://www...
...	...	..

Figura 3.1 - Visión relacional de la WWW.

estructura del grafo y tipos de nodos. Por ejemplo [Parunak91] combina un motor de hipermedia con bases de datos relacionales, y presenta la información vía atributos atómicos, los cuales pueden ser consultados por los mecanismos provistos por el RDBMS. [Marmann92] propone el uso de una base de datos orientada a objetos como soporte para una aplicación hipermedial. En este caso el acceso a la información se logra usando el lenguaje de consultas provisto por dicha base de datos.

La WWW puede verse como una gran hipermedia donde los nodos son todos del mismo tipo, documentos hipermediales. Se están estudiando lenguajes para realizar consultas sobre este gran hipertexto. En [Mihaila96] se define un lenguaje que permite ubicar aquellas páginas que contengan algún texto en particular o la distancia (medida en links) a alguna página que contenga dicho texto. El lenguaje de alto nivel en este caso permite extraer información desde la Web tomando las múltiples ventajas de los múltiples servidores de índices existentes en la red, en forma totalmente transparente al usuario. La idea utilizada es darle a la Web una visión de base de datos relacional donde hay dos grandes tablas de documentos y de links (Figura 3.1) y a partir de allí realizar las consultas.

Document

URL	Title	Text	Length	Type	Modif
http://www...	Title1	Text1	1236	Test	1-1-96
...	...	...	...	...	...

Anchor

Base	Label	Href
http://www...	Label	http://www...
...	...	..

Figura 3.1 - Visión relacional de la WWW.

Capítulo 4
Modelo de Datos y definición de OOHQL

4.1 Introducción.

En este capítulo será presentado el lenguaje de consultas OOHQL (*Object Oriented Hypermedia Query Languaje*) [Gordillo98]. Se abarca la especificación del modelo de datos subyacente y de la sintaxis con la cual se expresa una consulta, proveyendo ejemplos de la utilización del mismo. Además serán discutidas las dos utilidades del lenguaje de consultas, estas son: la posibilidad de consultar la información contenida en el hipermedia, y la posibilidad de consultar información a cerca de la estructura del hipermedia.

Este capítulo esta organizado de la siguiente forma: en la sección 4.2 se presenta el modelo de datos subyacente al lenguaje de consulta para hipermedias. En la sección 4.3 se define el lenguaje de consulta, tanto para consultas de la información contenida en el hipermedia como para consultas del esquema. Se presentan además diversos ejemplos ilustrativos.

4.2. Modelo de datos.

Antes de definir un lenguaje de consultas en particular, debe ser definido el modelo de datos sobre el cual las consultas serán realizadas. Este modelo indica como se encuentra estructurada la información a ser consultada. Todo lenguaje de consultas esta relacionado con un modelo de datos subyacente. Por ejemplo, una consulta expresada en SQL hace referencias a relaciones, que son el elemento básico del modelo de datos relacional; en OQL se hace referencia a objetos debido a que su modelo de datos esta conformado por objetos. Las sintaxis permitidas por estos lenguajes están relacionadas con las restricciones impuestas por el modelo de datos.

4.2.1. Diseño de Hipermedias con un Modelo Orientado a Objetos.

El modelo de datos orientado a objetos es muy expresivo y permite un alto grado de reuso y extensibilidad. Como resultado de esto, es utilizado en una gran cantidad de aplicaciones en nuestros días. Definir un modelo de diseño de hipermedias no es tarea fácil, el modelo debe permitir organizar la información y preservar las facilidades de las aplicaciones hipermediales como son la navegación, poseer diferentes perspectivas de la misma información, interfaces multimediales, etc.

Se utiliza la potencia de los modelos orientados a objetos para la definición de aplicaciones hipermediales.

El modelo que se propone consiste básicamente de dos niveles:

Diseño de alto nivel

El objetivo es diseñar la aplicación desde un punto de vista abstracto, dejando de lado los detalles y concentrándose en los elementos más importantes y las relaciones existentes entre ellos. El mecanismo de diseño es similar al propuesto por Rumbaugh en [Rumbaugh91], el cual presenta las clases de la aplicación organizadas en jerarquías de especialización y composición, y las relaciones entre ellas. En este nivel sólo son descriptos los objetos de aplicación y sus relaciones, junto con sus propiedades y comportamiento, pero no es definida la información a cerca de las características de navegación, interfaces de objetos y detalles de implementación.

Entidades: cada tipo de entidad en el dominio, es modelada como un tipo de nodo, y es definido por un conjunto de propiedades y por su comportamiento. Estos nodos pueden estar relacionados por las relaciones de subclase (is_a) y de composición (part_of).

Un atributo (propiedad) está definido por un nombre y un dominio. Los atributos son clasificados de acuerdo a su dominio en las siguientes siete categorías:

- *Atributo atómico:* su dominio es una clase primitiva (en este modelo pueden ser texto, sonido, video, etc.).
- *Atributo complejo:* toman valores en clases definidas por el usuario.
- *Atributo compuesto:* su dominio es una clase no primitiva y soporta la semántica de la relación part_of.
- *Atributo simple:* contiene un solo valor del dominio especificado.
- *Atributo multi-valuado:* es definido por los constructores set_of o list_of, contiene varios valores del dominio especificado.
- *Atributo heterogéneo:* puede contener valores de varios dominios diferentes.
- *Atributos Derivados:* Toma su valor en tiempo de ejecución desde una función.

Hay una clase de nodo que es la raíz de la jerarquía *is_a*.

Relaciones: Las relaciones del modelo conceptual relacionan dos o más entidades de la aplicación. Desde el punto de vista de una hipermedia, las relaciones definen links para navegar las aplicaciones hipermediales.

Las relaciones de este modelo son objetos y esta organizados en jerarquías de especialización/generalización, donde la raíz es una clase llamada *Link*.

La clase *Link* define algunos atributos para especificar el nodo fuente, el nodo destino, cardinalidad y comportamiento para la navegación.

Una relación de aplicación hereda los atributos de la clase *Link* y probablemente agregue sus propios atributos.

Diseño de bajo Nivel

En este nivel el objetivo es:

- Diseñar las estructuras de navegación para definir anclas (*anchor*, es la palabra de origen ingles utilizada por todas las literaturas sobre el tema para nombrar a las anclas).
- Definición de perspectivas para la visualización de la información.
- Descripción de los detalles de las interfaces de usuario.

Una interesante diferencia entre las aplicaciones de bases de datos y aplicaciones hipermediales, es que en estas últimas los objetos pueden tener diferentes representaciones porque los lectores pueden observar la hipermedia desde diferentes perspectivas. Con el objetivo de permitir esta capacidad, este modelo permite que diferentes vistas sean definidas sobre la información contenida en un nodo. Se utiliza para tal propósito el concepto de *Exemplar* definido en [Lalonde89].

Exemplar: está definido por una lista de atributos (estos atributos son una selección de los existentes en la clase de nodo asociada) y también por el comportamiento esperado. Cada clase de nodo tiene al menos un exemplar asociado. Un exemplar contiene anclas correspondientes a los links seleccionados para visualizar la perspectiva definida por el exemplar.

En el mismo camino que los nodos, el exemplar se organiza en jerarquías de especialización donde la raíz es la clase Exemplar.

Los niveles previamente descriptos definen el esquema del modelo, en otras palabras, el esquema está compuesto por tres jerarquías: clases de nodos describiendo las entidades de la aplicación, clases de links con descripción de relaciones y clases exemplar que definen perspectivas y anclas.

Este esquema es la base del lenguaje de consultas definido. Siguiendo la idea de la introducción, tenemos que SQL referencia relaciones, OQL referencia objetos, y OOHQL referencia nodos y links.

4.2.2. Un ejemplo.

En esta sección presentaremos un ejemplo que será utilizado durante el desarrollo de esta tesis como medio de ejemplificación de conceptos. El ejemplo completo puede apreciarse en el apéndice A. La aplicación modelada es una hipermedia de una librería (Figura 4.1). Se utiliza una notación UML-Like [Rational97], donde las cajas representan nodos, las flechas links y las flechas con rombos una relación de composición. La letra negra en las cajas representa el nombre del nodo y las letras de tipografía diferente dentro de la caja representa datos que estos nodos manipulan. Los nombres asociados a las flechas determinan el nombre del link.

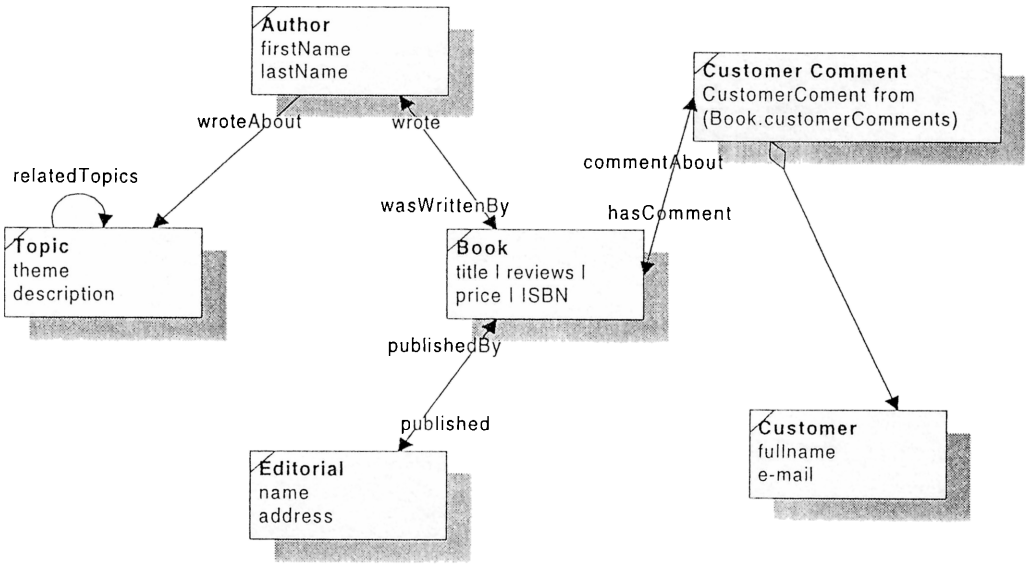


Figura 4.1 - Diagrama de clases de la aplicación.

4.3 OOHQL (*Object Oriented Hypermedia Query Language*)

Este lenguaje combina las estructuras de las bases de datos orientadas a objetos y la navegación de las aplicaciones hipermediales. El modelo de consulta está basado en el modelo definido en [Kim90], y sobre el cual se han hecho algunos cambios para trabajar con aplicaciones hipermediales. El lenguaje permite recuperar información en dos sentidos: recuperar información contenida en el hipermedia y consultar información del esquema de alguno de los árboles de jerarquía (clases *Node, Link y Exemplar*).

4.3.1. Tipos de nodos y tipos de links

En nuestro trabajo los términos Clases y Tipos tienen diferentes significados. Mientras que una clase define la especificación para un conjunto de objetos (es decir, define un conjunto de instancias pertenecientes a una clase); un tipo define el protocolo de un grupo de objetos. Como consecuencia de esto, el esquema de tipo y el esquema de tipos y el esquema el esquema de clases podría ser diferente. Por ejemplo, nosotros tenemos dos clases que son *Capitán* y *Personal*, ambas pertenecen a un mismo tipo Persona (responden a un mismo protocolo) y sin embargo fueron definidos por diferentes clases.

Los dos mecanismos coexisten en el sistema, son independientes entre si, no es necesario que compartan ambas clase una misma superclase para pertenecer al mismo tipo.

Nuestro trabajo utiliza extensiones de tipo (nodos y links que cumplen con un tipo dado) y no con extensiones de clases (nodos y links pertenecientes a la misma clase)

4.3.2. Consulta de Información.

Estas consultas están orientadas al usuario, permiten la recuperación de la información almacenada en el hipermedia, y el operador más importante es la selección.

Selección. Esta operación selecciona objetos desde uno o más tipos, es similar a la selección en bases de datos relacionales.

select [targetClause]
    from rangeClauses
    [*in* source]
    [*where* qualificationClause]

donde:
targetClause: es la especificación de los tipos de salida.
rangeClause: indica la ligadura de los tipos con sus correspondientes conjuntos de instancias.
source: es el hipermedia o el conjunto de nodos que pueden ser consultados.
qualificationClause: es una combinación de predicados booleanos.

[...]: indica parámetros opcionales.

El resultado de una consulta se conforma con una lista de instancias de nodos que cumplen el predicado de consulta, más las relaciones existentes entre ellos. En otras palabras, es un sub-conjunto de nodos del hipermedia original, pero tiene solo aquellas

instancias de los tipos involucrados en el *targetClause* que cumplieron con la *qualificationClause*, más todas las relaciones entre ellos.

Por medio de la *asignación* (':='), se puede almacenar el resultado de una consulta en una variable. Esta capacidad permite realizar consultas usando como fuente de información el resultado de una consulta anterior.

Ejemplo.1) Seleccionar todos los autores con nombre Orfali Robert, donde Q es una variable para almacenar el resultado.

```
Q : = SELECT Author
      FROM  _a: Author
      WHERE  (_a name = 'Orfali Robert')
```

Related_by operation. El funcionamiento del operador Related_by permite establecer si hay una relación definida por un link entre dos objetos nodo. El resultado de esta operación es un valor booleano, verdadero en caso de que exista tal relación, y falso en caso contrario.

La sintaxis de esta consulta es:

Related_by (*objetoFuente, tipo de Link ,objetoDestino*)

Ejemplo.2) Seleccionar todos los Autores del libro "Java with Corba".

```
SELECT  Author
FROM  _b:Book , _a: Author
WHERE (_b title = 'Java with Corba') AND
          (Related_by(_b,wasWrittenBy,_a))
```

Path. La operación path permite establecer si hay un camino de relaciones entre objetos nodo. El resultado de esta operación es un valor booleano, verdadero si el camino existe, y falso en caso contrario.

La sintaxis de la función es:

Path(*objetoFuente, lista de Tipos de Link, objetoDestino*)

Ejemplo.3) Seleccionar todos los autores que tienen libros publicados en la editorial Wiley.

```
SELECT Author
FROM  _a:Author , _e:Editorial
WHERE  (_e name = 'Wiley') and (Path(_a,wrote,publishedBy,_e))
```

Cuantificadores. La *qualificationClause* puede incluir atributos multivaluados. Por esta razón, el modelo incluye dos cuantificadores: EACH y EXIST, correspondiente al cuantificador universal y existencial respectivamente.

Ejemplo.4) Seleccionar todos los libros en los cuales Abiteboul sea el autor.

```
SELECT  Book
FROM  _b:Book , _a:Author
WHERE  (_a name = 'Abiteboul Serge') AND
          (EXIST Related_by(_b,wasWrittenBy,_a))
```

Projection. La operación de proyección permite recuperar atributos a través de mensajes desde objetos de uno o más tipos. Esta operación es similar al operador de proyección de sistemas de bases de datos relacionales.

PROJECT [targetClause]
from rangeClauses [in source]
[where qualificationClause]

donde la definición de *rangeClause*, *source*, y *qualificationClause* son las mismas que para la operación *select* descripta anteriormente. El *targetClause* ahora está formado por una lista de atributos. Los atributos son referenciados con la notación:
Tipo mensaje, donde mensaje es el método que se le debe invocar a cada instancia perteneciente a ese Tipo para obtener el atributo esperado en el resultado.

La respuesta a una consulta de proyección es una lista de los atributos especificados.

Ejemplo.5) Titulo de los libros junto al nombre de la editorial que lo edita.

```
PROJECT  Book title, Editorial name
FROM  _b:Book , e:Editorial
WHERE  (Related_by(_b,publishedBy,_e))
```

IsPartOf. Este operador permite recuperar partes de una instancia.

Ejemplo.6) Todos los libros que tengan un comentario de 'Marcos Guevers'.

```
SELECT Book
FROM _b:Book , _c:Customer , _cc:CustomerComment
WHERE (_c fullName = 'Marcos Guevers') AND
          (Related_by(_b, hasComment, _cc)) AND
          (_c is_part_of _cc)
```

Navegación del resultado. El objetivo es poder navegar a través del hipermedia resultante. Con este propósito, se proponen diferentes maneras para organizar los nodos del resultado: una secuencia de nodos, una secuencia ordenada de nodos accedida a través de un índice, un conjunto de nodos indexados y un conjunto de nodos con acceso random.

Para poder crear estas estrategias, se encuentran definidos los siguientes constructores: *list*, *set* y *index_by*. Mientras el operador de lista define una secuencia de nodos unidos por una condición especifica, el operador de conjuntos define un conjunto de nodos desconectados, y *index_by* define acceso directo a nodos ordenados secuencialmente o nodos desconectados. En el siguiente cuadro se muestra como se especifican las posibles estrategias de navegación sobre el resultado de una consulta Q:

C := *set (Q)*	Define un conjunto de nodos con acceso random
C := *set (Q) indexed_by* selector	Define un conjunto de nodos desconectados pero con algún índice
C := *list (Q) sorted_by* selector	Define una secuencia de nodos
C := *list (Q) sorted_by* selector *indexed_by* selector	Define una secuencia de nodo con un índice

4.3.3. Consultas sobre el esquema.

Este tipo de consulta es útil para el diseñador del hipermedia, le permite obtener información sobre cualquiera de las jerarquías de esquema del hipermedia.

Hierarchy. Esta operación se usa para establecer cual es la superclase o subclase de una clase dada.

Hierarchy ↑ *claseName* ***From*** *Hierachy_Type*

or

Hierarchy ↓ *claseName* ***From*** *Hierachy_Type*

donde:

className: es un nombre de clase de cualquier jerarquía.

hierachy_type: Es alguna jerarquía de *Nodo, Link o Exemplar.*

La diferencia entre *hierarchy*↑ y *hierarchy*↓ es que la primera retorna el nombre de la superclase y la segunda los nombres de las subclases.

Properties. Recupera las propiedades definidas en la jerarquía.

properties *className* ***from*** *hierarchy_type*

className y *hierarchy_type* son definidas como en la operación *hierarchy.*

Source and target. Recupera la clase del nodo fuente y el destino de una relación específica.

source *className*

target *className*

donde:
className es el nombre de una clase de la jerarquía de *Links*.

Exemplar. Esta consulta retorna el nombre de los exemplars asociados a una clase de nodo específica.

exemplar *className*

donde:
className es el nombre de la clase de *Nodo*.

Related_to y Related_from. La operación *related_to* determina las clases de nodos que son alcanzadas desde una clase específica, y la operación *related_from* define todos las clases de nodos desde la cual una clase específica puede ser alcanzada.

related_to *nodeClass*

related_from *nodeClass*

donde:
nodeClass es el nombre de la clase de *Nodo*.

Capítulo 5

Herramientas Complementarias

5.1. Introducción

OOHQL se define sobre un modelo de datos abstracto. Dicho modelo fue presentado en el Capítulo 4 de esta tesis, y es sumamente importante, ya que está estrechamente relacionado con el lenguaje de consulta.

El primer paso en dirección a obtener una implementación de un motor de consultas basado en OOHQL, fue el estudio de las posibles herramientas de implementación de una aplicación hipermedial. En este sentido, nos abocamos a determinar cuales son las herramientas conceptuales y tecnológicas involucradas en el proceso de desarrollo de dicha aplicación. Cabe destacar en este punto, que a partir de aquí, cada vez que en este trabajo se haga referencia a la aplicación hipermedial, nos estamos refiriendo a una aplicación hipermedial orientada a objetos, es decir, un hipermedia desarrollada con tecnología de objetos.

OOHDM es una metodología de diseño para el desarrollo de aplicaciones hipermediales orientadas a objetos. La aplicación de dicha metodología arroja como resultado un modelo de datos similar al definido junto con OOHQL.

OO-Navigator [Garrido96] [Garrido97] es un framework orientado a objetos que permite el desarrollo de aplicaciones hipermediales orientadas a objetos. Existe una implementación de dicho framework realizada en el lenguaje de programación Smalltalk [Goldberg83] y permite instanciar de manera natural aplicaciones hipermediales diseñadas con OOHDM.

Utilizamos OOHDM como metodología de diseño de nuestra aplicación hipermedial, implementamos dicha aplicación utilizando el OO-Navigator Framework y montamos el lenguaje de consultas sobre dicha implementación.

En este capítulo se presenta la relación existente entre estas tecnologías y como se relaciona nuestro trabajo con ellas. En la sección 5.2 analizamos brevemente la metodología de diseño de aplicaciones hipermediales OOHDM. En la sección 5.3 definimos conceptualmente qué es un framework y presentamos el OO-Navigator. Finalmente, en la sección 5.4 explicamos como se integran estas dos herramientas tecnológicas con OOHQL.

5.2. OOHDM (Object-Oriented Hypermedia Design Methodology)

En esta sección se presenta la metodología de diseño de aplicaciones hipermediales a utilizar. Para obtener mayor información sobre esta metodología de diseño y su utilización, el lector puede remitirse a los artículos [Schwabe95] y [Schwabe96].

OOHDM define cuatro actividades en el proceso de diseño de una aplicación hipermedial orientada a objetos. Luego de las tres primeras actividades se obtiene un modelo de diseño de una parte de la aplicación, y durante la cuarta actividad definida en OOHDM se implementa cada uno de estos modelos.

La primera actividad definida en OOHDM se denomina Diseño Conceptual. En esta etapa se diseña la aplicación orientada a objetos sobre la cual se basará la aplicación hipermedial. Esta aplicación de objetos, encierra la lógica referente al dominio de la aplicación. Es una aplicación convencional orientada a objetos sin ningún tipo de característica hipermedial.

La segunda actividad se denomina Diseño Navegacional. En ella el diseñador adiciona características hipermediales a la aplicación orientada a objetos diseñada previamente, definiendo nodos y links que mapean a objetos y relaciones entre objetos provenientes de la aplicación subyacente. Durante esta actividad se define la forma en que se navegará la información en la aplicación hipermedial.

La tercera actividad se denomina Diseño Abstracto de Interfaces, donde se define la forma en que los nodos de la aplicación hipermedial definida anteriormente, serán visualizados. Cada nodo puede tener diversas vistas dependiendo del contexto de usuario en el cual esté siendo navegado.

Finalmente, la cuarta actividad consiste en la implementación de cada uno de los modelos obtenidos anteriormente, utilizando alguna herramienta de desarrollo de aplicaciones hipermediales adecuada. Como resultado se obtiene una aplicación hipermedial ejecutable sobre la que posteriormente se realizarán las consultas.

La Tabla 5.1 muestra una descripción resumida de las actividades involucradas en el proceso de desarrollo de una aplicación hipermedial utilizando OOHDM.

Etapas	Productos	Mecanismos	Diseños Concernientes
Diseño Conceptual	Clases, subsistemas, relaciones, perspectivas de atributos.	Clasificación, composición, generalización, y especificación.	Modelar la semántica del dominio de aplicación.
Diseño Navegacional	Nodos, links, estructuras de acceso, contextos navegacionales.	Mapeo entre el modelo conceptual y los objetos navegacionales.	Enfasis sobre aspectos cognitivos. Toma en cuenta perfiles de usuario y tareas.
Diseño abstracto de Interface	Objetos abstractos de interface, responsabilidades de eventos externos, transformación de interfaces.	Mapeo entre objetos navegacionales y objetos perceptibles.	Modelo de objetos que se pueden percibir, elegir metáfora de implementación. Describe interface para objetos navegacionales.
Implementa-ción	Ejecutar la aplicación.	Proveer un medioambiente.	Performance, completitud.

Tabla 5.1. Resumen de la metodología OOHDM

5.3. OO-Navigator Framework

5.3.1. Frameworks orientados a objetos

Un fenómeno ampliamente conocido por la comunidad de personas que se dedican a la construcción de sistemas de software, es el hecho de que los sistemas de software tienden a crecer con el tiempo, a medida que se le agregan nuevas características (muchas veces como consecuencia de un cambio en los requerimientos de los usuarios), se modifica la funcionalidad de algunas de sus partes, y se reemplazan partes antiguas. Todo esto se traduce en que se re-escribe código, y se extiende código ("a la defensiva"), agregando nuevas versiones sin modificar las versiones anteriores, en lugar de actualizar la versión corriente. El problema surge cuando el diseño no era el adecuado para los propósitos inherentes al sistema. Esto resulta en que la estructura de un programa se deteriora con el tiempo.

La única manera de evitar esta decadencia en la estructura de un programa que debe mantenerse es, según Opdyke y Johnson [Opdyke90], re-escribiéndolo de manera tal de ir abstrayendo y mejorando su estructura.

La orientación a objetos se promociona como la mejor forma de reusar y extender programas, pero la estructura de un programa OO también se deteriora a medida que se agregan nuevas interacciones y se implementan nuevos requerimientos. Las jerarquías se hacen grandes y con menos sentido, se duplica código y las clases individuales se hacen grandes y difíciles de entender y manejar.

La manera de resolver esta situación es re-factorizando código de tiempo en tiempo. Re-factorizar un programa orientado a objetos implica ir creando superclases e ir trasladando hacia ellas el comportamiento común a las subclases, y así obtener clases abstractas.

Un *framework,* en el campo de la orientación a objetos, es un diseño arquitectónico de un tipo particular de aplicación o dominio [Johnson88]. Consiste en un conjunto de *clases abstractas y concretas*, y define la *interacción* entre los componentes mediante el planteo de restricciones, herencia, polimorfismo y reglas informales de composición.

Las *clases abstractas* conforman los componentes principales de la arquitectura de un framework, y por lo tanto representan el esqueleto que va a conservar toda aplicación del dominio que el framework representa. El flujo de control que se especifique a través de las interacciones entre componentes, deberá capturar las decisiones de diseño comunes en ese dominio.

Un framework tendrá distintas *instancias* de sí mismo en el dominio para el cual fue planteado, lo que implica que su modelo debe ser lo suficientemente general para cumplir con su objetivo fundamental: el *reuso*, no sólo de código sino también de diseño. Cada instancia particular será definida por las *clases concretas* que reemplazarán a las abstractas en tiempo de ejecución, usando los mecanismos de herencia y polimorfismo. El usuario del framework podrá elegir las clases concretas que necesite entre aquellas provistas por el framework en forma de biblioteca, o crearlas por sí mismo. Esto significa que el diseño debe ser *extensible,* es decir, tener una estructura abierta a nuevas subclases. Además, su interface deberá especificar cómo se relacionará con el resto del sistema.

De lo anterior podemos deducir que la construcción de frameworks OO es muy difícil. Requiere un conocimiento profundo tanto del dominio a ser modelado, como del perfil de aplicaciones que podrán hacer uso del mismo. No existen técnicas formales de

desarrollo de frameworks hasta el momento, pero nuevos adelantos se están produciendo para permitir una buena documentación de los mismos.

Los beneficios de usar frameworks son considerables, y de ahí que se sigan desarrollando a pesar de su alto costo: construcción más veloz de aplicaciones, y fácil mantenimiento, debido a que todas las aplicaciones que hagan uso de un framework particular exhibirán una estructura de diseño común, y a que sólo se necesita elegir las clases concretas rellenando ese esqueleto, para instanciarlo.

5.3.2. El OO-Navigator Framework.

OO-Navigator es un framework orientado a objetos para la construcción de aplicaciones hipermediales, o extensión de aplicaciones orientadas a objetos con funcionalidad hipermedial.

Mezclar los aspectos de interface y navegación dentro de la aplicación a ser extendida derivaría en un código oscuro, inmantenible y poco o nada reusable. Uno de los principios de la reusabilidad de grandes arquitecturas es la separación de las mismas en capas independientes y cohesivas en sí mismas. Por esto, OO-Navigator ha definido tres niveles de abstracción para cumplir con su objetivo: el nivel de objetos, el nivel de hipermedia y el nivel visual.

A continuación describimos cada nivel.

Nivel de objetos. Es el nivel de la aplicación subyacente, es decir que estará formado por los componentes involucrados en el modelo de esta aplicación. Esta capa será la responsable de proveer la información a ser mostrada, las relaciones a ser navegadas, y el comportamiento a ser extraído por el nivel de hipermedia.

Si el usuario del framework estuviera creando una aplicación hipermedia desde el principio, debería modelar previamente el dominio, diseñando este nivel de la misma forma que propone OOHDM en su actividad de diseño del modelo conceptual.

Nivel de hipermedia. Estos componentes modelan e implementan en la forma más general posible los conceptos que definen una aplicación hipermedia. El usuario los instanciará en el momento de definir la visión navegacional sobre el nivel de objetos.

OOHDM propone una segunda actividad, diseño navegacional, en la que el diseñador define las clases de componentes navegacionales que mapean las del modelo conceptual, lo que se traduciría a instanciar las clases que sean definidas en este nivel.

Nivel visual. Aquí se manejan los aspectos de interface, es decir, se especifica la apariencia visual de cada componente del nivel anterior, y se controla la interacción con el mundo exterior.

El OO-Navigator utiliza otro framework para este nivel: el "Model View Controller" (MVC) [Krasner88], el cual se ha extendido para soportar objetos de interface con capacidad de navegación. A pesar de que el framework de hipermedia es independiente de este nivel y podría ser usado con otra metáfora de interface, utiliza el MVC por estar siendo referenciado como el mejor modelo de interface actual, y ser prevaleciente en los sistemas OO.

OOHDM, así como la mayoría de los métodos de diseño de hipermedias, propone actividades de desarrollo que bien podrían ser implementadas con cada nivel de esta arquitectura.

La metodología OOHDM también separa los aspectos de interface en una tercera actividad llamada diseño abstracto de interface.

El framework de hipermedia involucra los componentes del segundo nivel y también la comunicación necesaria entre él y los restantes niveles.

En la Figura 5.2 puede verse gráficamente esta división por capas del OO-Navigator.

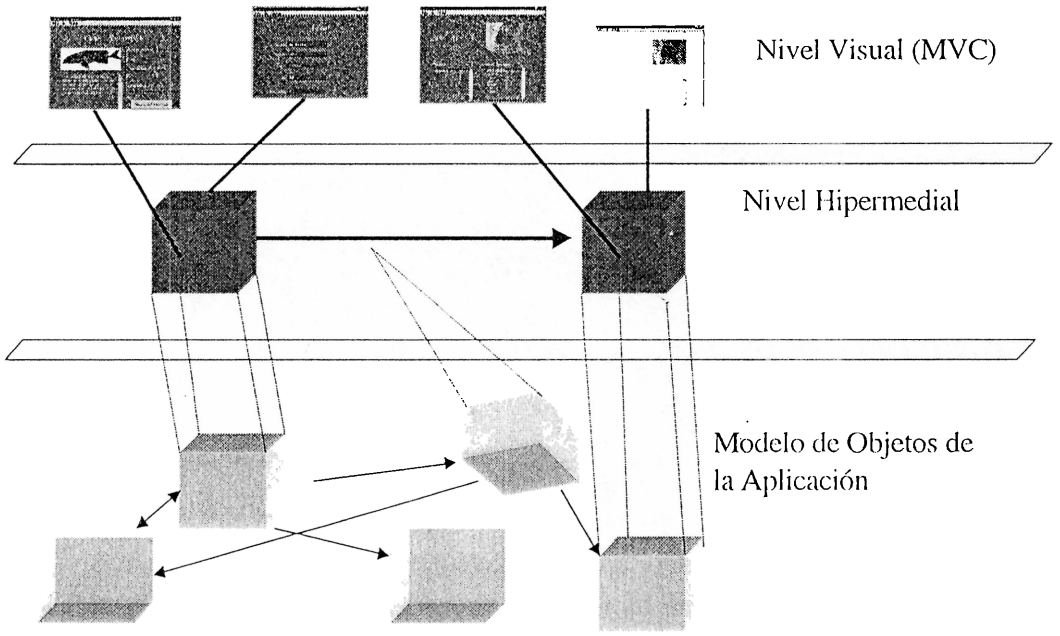


Figura 5.2. División por capas del OO-Navigator Framework

Como puede apreciarse en la Figura 1, no todo objeto del modelo de aplicación necesariamente debe tener una representación a nivel hipermedial. A sí mismo, algunos objetos se pueden mapear como nodos a nivel hipermedial y otros se pueden mapear como links. Cada nodo puede tener una o más vistas definidas, un nodo puede verse diferente si el hipermedia es observada, por ejemplo, por un cliente o el dueño de la empresa.

Las consultas se realizan a nivel hipermedial y el resultado de la consulta es un subconjunto de instancias de ese nivel.

5.4. Una aproximación sistemática para el procesamiento de consulta en aplicaciones hipermediales.

Para poder ejecutar consultas sobre una aplicación hipermedial es necesario tener un esquema que describa la información contenida en dicha aplicación. Esto es análogo a lo que sucede en las bases de datos relacionales, para consultar información es necesario conocer al menos, un bosquejo del diagrama de entidad-relación de la base de datos. Para definir este esquema, el trabajo ha sido basado en la metodología OOHDM, esta metodología comprende cuatro etapas para el desarrollo de una aplicación hipermedial (dichas etapas fueron descriptas anteriormente). Nosotros usamos los modelos obtenidos de las dos primeras etapas y agregamos la especificación de una tercera para la ejecución de consultas. Nuestra arquitectura define tres capas de acuerdo a este modelo.

La primer capa es llamada modelo de objetos o aplicación orientada a objetos. Esta contiene la fuente de datos para la aplicación hipermedial. Esta capa es equivalente al modelo conceptual en OOHDM, donde un modelo de objetos es construido usando los principios del paradigma orientados a objetos.

La segunda capa es llamada aplicación hipermedial. En OOHDM una aplicación hipermedial es concebida como una vista navegacional por encima del modelo conceptual. Esta aplicación se construye durante la actividad de Diseño Navegacional, donde se definen nodos y links como vistas navegacional de objetos en el modelo de objetos.

La tercera capa está compuesta por lo que denominamos el QSC (*Query Solving Component*) que implementa el lenguaje de consulta OOHQL y provee las capacidades de consulta sobre una aplicación hipermedial orientada a objetos.

En la Figura 5.3 se muestra el esquema resultante para esta arquitectura de tres capas.

Las dos primeras capas observadas en la Figura 5.3 (la aplicación orientada a objetos y la aplicación hipermedial) están implementadas por el OO-Navigator Framework.

Integrando las ideas propuestas en OOHDM, con el OO-Navigantor y OOHQL, podemos dar pautas para un proceso sistemático de consultas sobre aplicaciones hipermediales:

- El primer paso consiste en el modelado la aplicación hipermedial utilizando OOHDM (modelando previamente una aplicación de objetos subyacente a la que se quiere dar funcionalidad hipermedial, o no).
- El segundo paso es la implementación del modelo usando el OO-Navigator como herramienta de desarrollo.
- Finalmente se provee la capacidad de realizar consultas sobre dicha aplicación usando OOHQL y según el modelo de navegación definido en el primer paso.

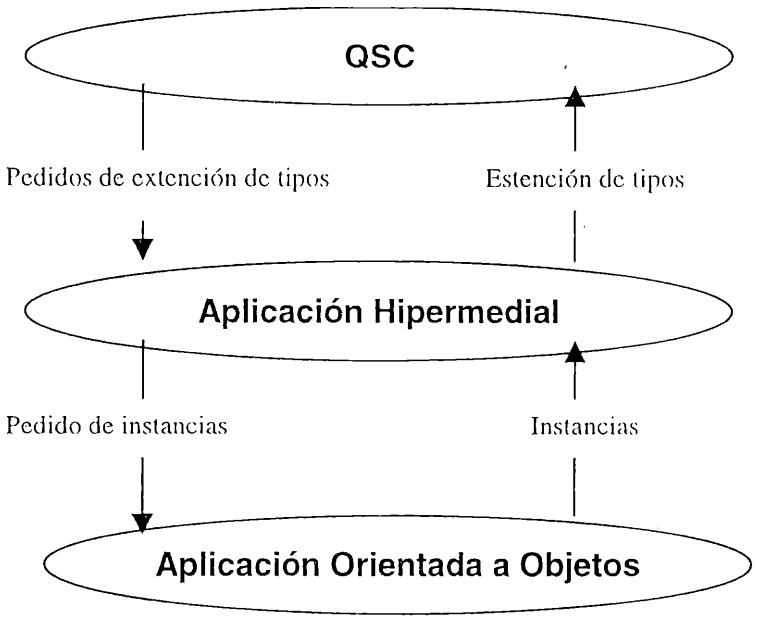


Figura 5.3. Arquitectura de tres capas.

La Figura 5.4, muestra el uso de estas herramientas en cada uno de las capas anteriores. En esta figura pueden observarse a la derecha las herramientas tecnológicas utilizadas en cada capa, y a la izquierda la aplicación o componente obtenidos.

Utilizando VisualWorks se implementa la *aplicación OO* que provee la *fuente de Información* para la aplicación hipermedial, y a la que luego, mediante la instanciación del OO-Navigator se aumenta con *funcionalidad hipermedial*, obteniendo una *aplicación hipermedial*. Finalmente, utilizando OOHQL como lenguaje de consulta se provee la posibilidad de realizar consultas sobre dicha aplicación, utilizando el QSC y "conectándolo" a la *aplicación hipermedial*.

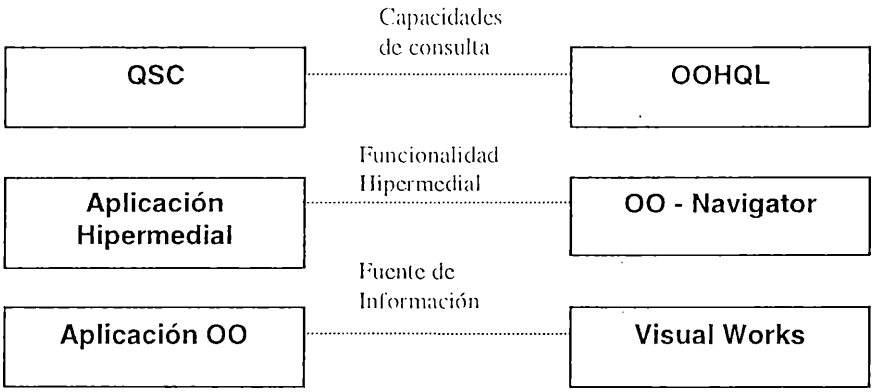


Figura 5.4. Integración de herramientas.

Capítulo 6
Un modelo algebraico para la implementación del QSC.

6.1 Introducción.

Como mencionamos anteriormente en el Capítulo 3, todo modelo de datos tiene asociado una representación formal. Así por ejemplo, el modelo de datos relacional tiene sus fundamentos en la teoría de las álgebras relacionales [Codd70]. No es el propósito de esta tesis discutir sobre las ventajas asociadas a la formalización de los modelos de datos, pero cabe destacar que la formalización permite el estudio teórico de los problemas derivados de la utilización de dichos modelos de datos. De tal forma, el análisis por ejemplo, de la complejidad en la resolución de consultas sobre bases de datos relacionales, es llevado a cabo mediante la representación algebraica de las consultas expresadas en algún lenguaje declarativo (como por ejemplo SQL). Esta representación algebraica es luego la base para el estudio de las posibles optimizaciones que pueden realizarse y del costo que pueden acarrear.

Si nos detenemos en el álgebra relacional de Codd, podemos decir sin temor a equivocarnos, que el conjunto de operadores relacionales que conforman dicho paradigma, es en si mismo un lenguaje de consultas sobre bases de datos relacionales, de la misma forma en que SQL es también un lenguaje de consultas sobre las mismas bases de datos. De esta forma, la ejecución de la expresión algebraica adecuada arroja como resultado un conjunto de relaciones que satisfacen el predicado de la consulta.

Sobre la base de lo expuesto anteriormente, y tomando en cuenta el modelo de datos asociado a las aplicaciones hipermediales orientadas a objetos definido y explicado en el Capítulo 5, hemos extendido la definición de álgebras relacionales dada por Codd, generando un nuevo conjunto de operadores algebraicos que nos permiten representar formalmente las consultas hechas sobre aplicaciones hipermediales orientadas a objetos. Este nuevo conjunto de operadores, denominados OOHA (Object Oriented Hypermedia Algebra) es independiente del lenguaje declarativo de alto nivel que se utilice para formular la consulta. Es decir, toda consulta realizada sobre este tipo de aplicaciones, puede ser transformada en una expresión OOHA equivalente, independientemente del lenguaje utilizado para expresarla. En este sentido, podemos decir que OOHA es también, al igual que las álgebras relacionales, un lenguaje de consultas en si mismo.

Este capítulo está organizado de la siguiente manera: en la sección 6.2 se justifica la elección de una aproximación algebraica para la representación de las consultas. En la sección 6.3 presentamos la definición completa y formal del álgebra por nosotros propuesta, ejemplificando su utilización y discutiendo algunos conceptos fundamentales de su definición. En la sección 6.4 analizamos el proceso de ejecución de una consulta.

6.2 Por qué una álgebra?

En la Sección 3.3, se comentó la existencia de dos aproximaciones o metodologías diferentes para la resolución de consultas. La primera es la basada en expresiones algebraicas y la segunda la basada en estimación de costos. En ambientes orientados a objetos, las estimaciones de costo se realizan construyendo un grafo de conocimiento de objetos que permite estimar el costo de resolver una consulta dada. Para poder implementar esta metodología, necesitamos conocer la estructura física de almacenamiento de los nodos de la aplicación hipermedial. Es decir, debemos conocer la forma en que la aplicación subyacente obtiene persistencia.

Nosotros no controlamos la recuperación física de los nodos, ya que ésta queda en manos del repositorio de información (hipermedia orientada a objetos). La única manera de hacer optimización es definir un álgebra con la cual se puedan transformar las consultas a expresiones semánticamente equivalentes, y luego aplicar reglas para la optimización lógica de estas expresiones. Es decir, utilizamos el primero de los dos métodos, pero solo la primer parte del método basada en reglas de transformación.

6.3 Definición del álgebra.

OOHQL es un lenguaje de consultas declarativo de alto nivel. Para implementar un sistema de software que permita procesar consultas escritas en OOHQL, necesitamos poder expresar formalmente dichas consultas.

En el campo de las bases de datos relacionales, se utilizan las álgebras relacionales de Codd [Codd70] para expresar consultas formalmente. El conjunto de operadores algebraicos definidos por Codd, se basan en la Teoría de Conjuntos [Halmos60]. Las relaciones son conjuntos de tuplas, las tuplas son conjuntos de atributos, los atributos pertenecen a un conjunto determinado de valores (también llamado *dominio* del atributo), y los operadores algebraicos relacionales (en especial los joins) son funciones que toman conjuntos como operandos y retornan conjuntos como resultados.

En el presente trabajo, hemos utilizado las ideas básicas de las álgebras relacionales, para definir una nueva álgebra orientada a conjuntos que nos permita especificar formalmente consultas sobre aplicaciones hipermediales orientadas a objetos.

Antes de presentar la especificación completa de OOHA, vamos a definir algunos conceptos básicos fundamentales, necesarios para comprender la definición de cada uno de los operadores algebraicos.

Todos los operadores OOHA (excepto por uno, la Identidad) toman *tablas* como operandos. Las *tablas* son conjuntos homogéneos de *tuplas*, ya que todas las *tuplas* pertenecientes a una dada *tabla* deben tener la misma estructura de tipos. Las *tuplas* pueden ser definidas como un conjunto ordenado de nodos (a partir de este punto, hacemos referencia a los nodos de una aplicación hipermedial), donde cada elemento en la *tupla* contiene un nodo de un determinado tipo. Como dijimos recientemente, cada *tupla* tiene su propia *estructura de tipos*, la cual establece que tipo corresponde a cada elemento de la *tupla*. Ya que las *tablas* son conjuntos homogéneos de *tuplas*, todas las tuplas pertenecientes a una misma tabla deben compartir la misma *estructura de tipos*.

De esta forma, vamos a hablar generalmente de la *estructura de tipos de una tabla*, haciendo referencia a la *estructura de tipos* de la *tuplas* que dicha *tabla* contiene.

Finalmente, dos o más operandos se dirá que son *compatibles* si tienen la misma *estructura de tipos*. Formalmente, e_1 y e_2 serán *compatibles* si y solo si: siendo N el número de campos en cada *estructura de tipos de ambas tablas*, $\forall\, i$: $1..N$, $e_1[i] = e_2[i]$ donde $e[i]$ referencia al tipo del campo i de la tabla e.

La Figura 6.1 resume los conceptos descriptos anteriormente.

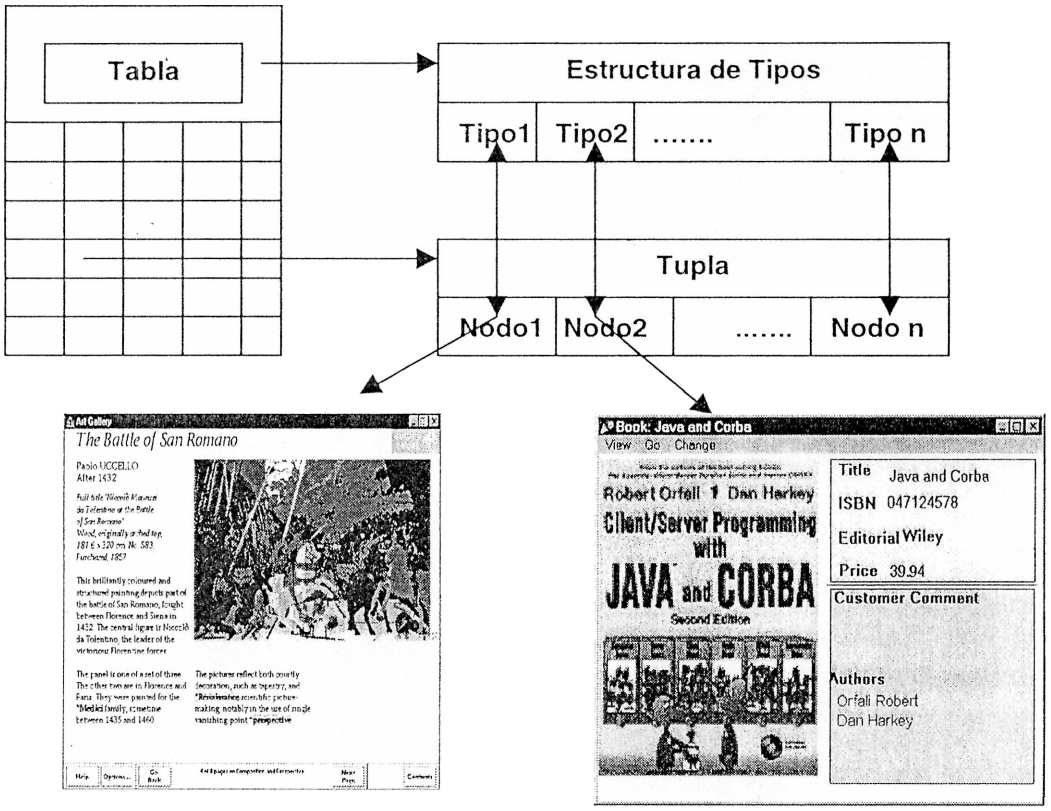


Figura 6.1 - Tablas, tuplas y estructuras de tipos.

6.3.1 Especificación de OOHA.

En las siguientes definiciones, e, e_1 y e_2 representan tablas; x, x_1 y x_2 representan tuplas; y $\{E \mid F\}$ es el conjunto de elementos E para los cuales se satisface el predicado F.
Para especificar los siguientes operadores algebraicos, tomamos las operaciones de Collect y Select de [Steenhagen96], y preservamos el estilo de notación de las restantes operaciones.

Collect: $\Gamma[x:f(x) \mid p(x)]\,(e) \equiv \{f(x) \mid x \in e \ \wedge p(x)\}$

Donde $p(x)$ es una expresión booleana evaluada sobre x, y f es una función. Este operador selecciona todos los objetos de e que satisfacen el predicado p y retorna una colección con los resultados de aplicar la función f a cada uno de ellos.

Select: $\sigma[x: p(x)](e) \equiv \{x \mid x \in e \wedge p(x)\}$

Donde $p(x)$ es una expresión booleana evaluada sobre x. Este operador selecciona todos los objetos de e que satisfacen el predicado p.

Attribute Projection: $\pi_{a1:T1,\ldots,an:Tn}(e) \equiv \{x[a_1:T_1,\ldots,a_n:T_n] \mid x \in e\}$

Donde $a_1:T_1$ referencia al atributo a_1 del nodo de tipo T_1. En este caso $x[a_1:T_1,\ldots,a_n:T_n]$ hace referencia a la tupla compuesta por los atributos $a_1:T_1,\ldots,a_n:T_n$ Correspondiente al elemento x del conjunto e. Debe notarse que, después de aplicar esta operación, ninguna otra operación puede ser aplicada sobre el resultado, ya que este resultado no es una tabla compuesta por nodos, sino por atributos de nodos.

Identity: $\gamma(type) \equiv \{[z] \mid z \in EXT(type)\}$

Donde $[z]$ es una tupla que contiene solo nodos pertenecientes a la extensión de tipo $EXT(type)$. Debemos recordar que este es el único operador OOHA que no recibe como argumento una tabla, sino un nombre de tipo *(type)*.

Object Projection: $T_1,..,T_n[e] \equiv \{x[T_1,..,T_n] \mid x \in e\}$

Donde $T_1,..,T_n$ es una lista de tipos de nodos. Es este caso $x[T_1,..,T_n]$ referencia la tupla compuesta por nodos de la tupla x pertenecientes al nodo del tipo T_1, luego al tipo T_2, y así sucesivamente.

Cartesian Product: $e_1 \times e_2 \equiv \{ x_1 \circ x_2 \mid x_1 \in e_1 \wedge x_2 \in e_2 \}$

Donde $\circ$ representa la concatenación de tuplas.

Regular Join: $e_1 \mid\bowtie_{x1,x2:p(x1,x2)} e_2 \equiv \{ x_1 \circ x_2 \mid x_1 \in e_1 \wedge x_2 \in e_2 \wedge p(x_1, x_2)\}$

Donde $p(x_1, x_2)$ es una expresión booleana evaluada sobre x_1 y x_2.

Union: $e_1 \cup e_2 \equiv \{x \mid x \in e_1 \vee x \in e_2\}$

Donde e_1 y e_2 deben ser compatibles.

Intersection: $e_1 \cap e_2 \equiv \{x \mid x \in e_1 \wedge x \in e_2\}$

Donde e_1 y e_2 deben ser compatibles.

Difference: $e_1 - e_2 \equiv \{x \mid x \in e_1 \wedge x \notin e_2\}$

Donde e_1 y e_2 deben ser compatibles.

Navigational Join:

$e_1 \oplus_{x1,x2:\,(x1[type1],\,L1,\ldots,Ln,\,x2[type2])} e_2 \equiv \{x_1 \circ x_2 \mid x_1 \in e_1 \wedge x_2 \in e_2 \wedge (x_1[type_1], L_1,..,L_n, x_2[type_2])\}$

Donde $(x_1[type_1], L_1,...,L_n, x_2[type_2])$ es una expresión booleana que involucra una comparación navegacional. Este predicado toma un nodo como origen. Este nodo pertenece a un tipo específico en una tupla específica $(x_1 [type_1])$. Luego, verifica la existencia de una camino de links pertenecientes a los tipos de link $L_1,..,L_n$ con destino $x_2 [type_2]$. En nuestro caso, este join representa a los operadores *Related_by* y *Path* definidos en OOHQL.

Composition Join:

$e_1 \otimes_{x1,x2\,:\,(x1[type],x2[type])} e_2 \equiv\{ x_1 \circ x_2 \mid x_1 \in e_1 \wedge x_2 \in e_2 \wedge (x_1[type], x_2[type]) \}$

Donde $(x_1[type], x_2[type])$ es una expresión booleana que representa la relación de "parte_de" que se verifica si y solo si x_2 "es parte de" x_1.

Natural Product:

$e_1 \bowtie e_2 \equiv \{f(x_1 \circ x_2) \mid x_1 \in e_1 \wedge x_2 \in e_2 \wedge p(x_1 ,x_2)\}$

Donde $p(x_1 ,x_2)$ es un predicado booleano que retorna true si todos los elementos de la tupla x_1 que tengan el mismo tipo que los elementos de la tupla x_2, son iguales. En caso contrario se retorna false.
Donde $f(x_1 \circ x_2)$ es una función que toma los elementos de $x_1 \circ x_2$ y elimina las repeticiones en las posiciones de la tupla que tengan el mismo tipo.

6.3.2 Ejemplos de expresiones OOHA bien formadas.

En esta sección resolveremos las consultas planteadas en el Apéndice A. Dichas consultas están expresadas en OOHQL, ahora expresaremos las mismas consultas pero utilizando expresiones OOHA.

Ejemplo.1) Seleccionar todos los autores con nombre Orfali Robert, donde q es una variable para almacenar el resultado.

$$\text{Author}[(\sigma_{[\,x \mid x[\text{Author}]\,\text{name} = \text{"Orfali Robert"}\,]}\ \gamma(\text{Author}))]$$

Ejemplo.2) Seleccionar todos los Autores del libro "Java with Corba".

$$\text{Author}[(\sigma_{[\,x \mid x[\text{Book}]\,\text{title} = \text{"Java and Corba"}\,]}\ \gamma(\text{Book})) \oplus_{x1,x2:\,(x1[\text{Book}],\,\text{wasWrittingBy},\,x2[\text{Author}])} (\gamma(\text{Author}))]$$

Ejemplo.3) Seleccionar todos los autores que tienen libros publicados en la editorial Wiley.

$$\text{Author}[(\gamma(\text{Author})) \oplus_{x1,x2:\ (x1[\text{Author}],\ \text{wrote},\ \text{publishedBy},\ x2[\text{Editorial}])} (\sigma_{[\ x\ |\ x[\text{Editorial}]\ \text{name} = \text{"Wiley"}\ ]}\ \gamma(\text{Editorial}))]$$

Ejemplo.4) Seleccionar todos los libros en los cuales Abiteboul sea el autor.

$$\text{Book}[\gamma(\text{Book}) \oplus_{x1,x2:\ (x1[\text{Book}],\ \text{wasWrittingBy},\ x2[\text{Author}])} (\sigma_{[\ x\ |\ x[\text{Author}]\ \text{name} = \text{"Abiteboul Serge"}]}\ (\gamma(\text{Author})))]$$

Ejemplo.5) Título de los libros junto al nombre de la editorial que lo edita.

$$\pi_{\text{title:Book,name:Editorial}}(\gamma(\text{Book}) \oplus_{x1,x2:\ (x1[\text{Book}],\ \text{publishedBy},\ x2[\text{Editorial}])}\ \gamma(\text{Editorial}))$$

Ejemplo.6) Todos los libros que tengan un comentario de 'Marcos Guevers'

$$\text{Book}[\gamma(\text{Book}) \oplus_{x1,x2:\ (x1[\text{Book}],\ \text{hasComment},\ x2[\text{CustomerComment}])} ((\sigma_{[\ x\ |\ x[\text{Customer}]\ \text{fullName} = \text{"Marcos Guevers"}]}\ \gamma(\text{Customer}) (\otimes_{x1,x2:\ (x1[\text{Customer}],\ x2[\text{CustomerComment}])}\ \gamma(\text{CustomerComment})))]$$

6.3.3 Orden de los operandos.

Una diferencia importante entre OOHA y las álgebras de Codd es la imposibilidad existente en las primeras, de conmutar el orden de los operandos en los joins. En las álgebras relacionales, si A y B son dos relaciones, entonces A *X* B y B *X* A son equivalentes, es decir, si bien lo pares ordenados que componen los dos resultados no serán los mismos (ya que los primeros serán de la forma $(tupla_A, tupla_B)$ y los segundos de la forma $(tupla_B, tupla_A)$), semánticamente ambas relaciones son equivalentes, representan a todas las tuplas de A relacionadas con todas las tuplas de B.
En OOHA esto no ocurre con todos los joins. En particular los joins de Navegación y de Composición no permiten conmutar sus operando. Esto es relativamente trivial, ya que por ejemplo, si tenemos el siguiente join de Navegación:

$$\oplus_{x1,x2:\ (x1[\text{Book}],\ \text{wasWrittenBy},\ x2[\text{Author}])}$$

Intercambiar x1 con x2 hace que la consulta deje de tener sentido, ya que posiblemente no exista un tipo de link "wasWrittenBy" que tenga como origen nodos del tipo Author y como destino nodos del tipo Book.
La situación es análoga para el join de Composición.

6.4 Procesamiento de consultas.

Antes de entrar de lleno en la definición de OOHA, que es el tema más importante de este capítulo, vamos a definir qué implica procesar una consulta sobre un repositorio de información.

Primeramente, vamos a adoptar como definición de "procesamiento de una consulta" al conjunto de pasos o etapas necesarias para satisfacer el requerimiento de información expresado por el usuario. En otras palabras, el procesamiento de una consulta comienza cuando el usuario escribe la consulta y no termina hasta que los resultados son debidamente retornados para que éste los utilice.

Este conjunto de pasos que hemos definido para procesar completamente una consulta, en el contexto de un hipermedia orientado a objetos como repositorio de información, se detalla a continuación, y está basado en el conjunto de pasos definido en [Oszu95].

- ***Creación de la consulta:*** la consulta se escribe utilizando un lenguaje de consulta declarativo de alto nivel, en nuestro caso OOHQL. EL usuario solo necesita conocer qué elementos de información puede consultar, esto es, que tipos de nodos y que tipos de links puede consultar.
- ***Normalización de la consulta:*** durante esta etapa la consulta es sometida a un proceso de "parseo" o análisis sintáctico que permite determinar si el usuario ha escrito la consulta siguiendo los lineamientos sintácticos del lenguaje en cuestión. Durante esta etapa no se realizan verificaciones de tipo antes de la ejecución de la consulta.
- ***Traducción algebraica:*** la consulta es traducida o transformada en una expresión algebraica (en efecto, una expresión OOHA) que combina componentes algebraicos que resuelven la consulta. El desacoplamiento del lenguaje de alto nivel que utilizan los usuarios para expresar las consultas, y la forma en que dicha expresión es transformada en una expresión algebraica equivalente, permite la evolución por separado de ambos lenguajes (el lenguaje de alto nivel declarativo y el álgebra).
- ***Optimización de la consulta:*** la expresión algebraica obtenida en el paso anterior es luego optimizada algebraicamente aplicando un conjunto de reglas de optimización. Cabe destacar que este proceso de optimización no contempla características físicas de almacenamiento que puedan ser aprovechadas para mejorar la performance de la ejecución de la consulta.
- ***Resolución del álgebra:*** en esta fase se ejecutan cada uno de los componentes algebraicos individuales que conforman la expresión optimizada resultante del paso anterior. Cada uno de estos componentes álgebraicos pueden ser implementados en diversas formas, y es en este punto donde sí pueden aprovecharse las características estructurales de la aplicación hipermedial para obtener mejores resultados en la ejecución de la consulta.

De esta forma, una consulta expresada por un usuario es sometida a cada uno de estos pasos en forma secuencial. El último paso, la resolución de la expresión OOHA que satisface la consulta, arroja como resultado el conjunto de nodos y links que satisfacen el predicado de consulta definido por el usuario.

Cabe destacar, que una vez obtenido el resultado, OOHQL establece diversas formas de accederlo. Pero esta característica es propia del lenguaje en cuestión, y puede no encontrarse presente en otros lenguajes de consulta sobre hipermedias alternativos. Es

por esto que no incluimos un sexto paso que de un formato particular al resultado, precisamente porque perderíamos generalidad.

6.4.1 Generación automática de la expresión OOHA.

Como definimos en el apartado anterior, el procesamiento de una consulta OOHQL comienza cuando, una vez que el usuario ha escrito la consulta, se somete al string que representa a esa consulta, a un proceso de análisis sintáctico dónde se determina si la consulta ha sido escrita correctamente, y donde, eventualmente, se normaliza dicha consulta eliminando condiciones redundantes y paréntesis innecesarios.

Ahora bien, el próximo paso en el procesamiento de una consulta OOHQL es traducir el string que representa a la consulta en una expresión OOHA correctamente formada. Este proceso de traducción algebraica involucra diversas etapas. Inicialmente, podría pensarse que es el analizador sintáctico o "parser" quien genera dicha expresión algebraica. Pero desafortunadamente, en esta etapa tan temprana del procesamiento de la consulta, el parser no tiene la información necesaria para poder crear una expresión OOHA correctamente formada.

Debido a lo expuesto anteriormente, necesitamos alguna forma de representación intermedia que nos permita, mediante un proceso posterior de análisis, generar la expresión algebraica. Es decir, debido a que el parser no cuenta con la información necesaria para crear la expresión OOHA, debe generar alguna representación intermedia de la consulta. Esta representación intermedia no es tan vaga como el string OOHQL, ni tan precisa como la expresión OOHA, pero permite a una tercera entidad, que por ahora denominaremos generador o "builder", construir la expresión OOHA final.

La Figura 6.2 esquematiza lo expuesto recientemente.

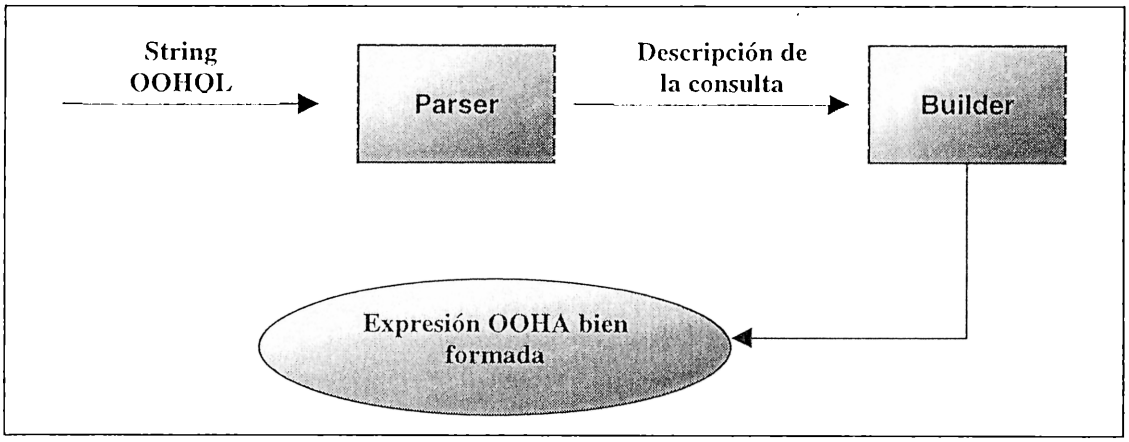


Figura 6.2 - Generación de la expresión OOHA.

De esta forma, el parser solo se encarga de la verificación sintáctica y el builder es quien genera la expresión algebraica. En la siguiente sección analizaremos más en detalle las responsabilidades de estos componentes, y las reglas que se aplican para generar la expresión OOHA bien formada.

6.4.2 Traducción, construcción y reglas de transformación.

Dada una consulta expresada en OOHQL existen varias posibles representaciones algebraicas que obtienen el resultado de dicha consulta. Si bien todas producen el mismo resultado, no todas son igualmente eficientes en ejecución. Esta eficiencia depende de diversos factores, tales como la velocidad de recuperación de instancias, la disponibilidad o no de índices, el orden de los operadores algebraicos, etc.

En esta tesis presentamos una forma particular de construir expresiones algebraicas, que no necesariamente genera la expresión óptima. El objetivo planteado es utilizar una forma de generación de expresiones algebraicas fácilmente extensible y mantenible.

Análisis sintáctico y Traducción: El Parser.

El parser, o analizador sintáctico, tiene la responsabilidad de verificar que el string OOHQL esté correctamente escrito, e interpretarlo y transformarlo en una representación intermedia con más información semántica que un string y sobre la que hay que aplicar luego diversas reglas de transformación para construir la expresión algebraica final.

El parser interpreta el string OOHQL y genera descriptores de operaciones a ser realizadas para obtener el resultado, pero no se encarga de combinar estos descriptores. Un descriptor determina cual es la operación que será realizada y cuales son los parámetros que necesita para ejecutar.

Por ejemplo, supongamos la siguiente consulta OOHQL:

```
SELECT Author
FROM  a: Author
WHERE   (a name = 'Orfali Robert')
```

Esta consulta OOHQL genera un descriptor de selección que usa la extensión de tipos de autor y debe cumplir la condición del *where*.

Si la consulta es más compleja, no alcanza solo con los descriptores. Hay que determinar exactamente cual es el tipo de relación existente entre ellos. El tipo de relación se refiere a que no son lo mismo dos operaciones unidas por un operador lógico AND que dos operaciones unidas por un operador lógico OR.

Por ejemplo, seria distinto tener en el *where* de la consulta anterior la siguiente expresión:

```
(a name = 'Robert Orfali') AND (a name = 'Serge Abiteboul')
```

que tener

```
(a name = 'Robert Orfali') OR (a name = 'Serge Abiteboul')
```

También hay que determinar el contexto en el cual se ejecuta la consulta, el contexto se refiere al significado de cada variable, un mismo nombre de variable puede tener asociado distintas extensiones de tipos en diferentes consultas.

También hay que determinar cual es la salida esperada de la consulta (desde el punto de vista de la estructura), la cual está determinada por la cláusula *select*. Puede ser una proyección de atributos o una proyección de objetos.

Construcción del álgebra: El Builder.

El builder posee una estructura capaz de interpretar y representar algebraicamente cada una de las cláusulas de una consulta OOHQL. La función del builder es generar los descriptores de cada operación y los repositorios que los contienen.

Un repositorio contiene un conjunto de elementos relacionados con el mismo criterio, donde por elementos nos referimos a descriptores o repositorios.

Por ejemplo, siendo A,B,C,D y E expresiones booleanas:

(A AND B)AND(C OR D)AND E => Repositorio $_{and}$ (E, Repositorio $_{and}$ (A,B), Repositorio $_{or}$ (S,D))

donde los subíndices indican el criterio de relación.

Los repositorios tienen un conjunto de reglas de generación de expresiones algebraicas, que determinan el criterio de relación anteriormente mencionado. Un repositorio de AND tiene reglas de generación asociadas a la conjunción, mientras que un repositorio de OR tiene reglas de generación asociadas a la disyunción.

Generación de los operadores OOHA.

Basamos la generación del álgebra en un mecanismo de aplicación sucesiva de reglas simples. Cada regla realiza solo un parte de la construcción. Mediante la aplicación sucesiva de un conjunto completo de reglas, en el orden adecuado, obtenemos la expresión algebraica deseada. La razón por la que elegimos esta metodología es la extensibilidad y simplicidad que provee.

Simplicidad ya que cada regla se encarga de algo muy especifico, para comprenderla solo debemos centrarnos en una actividad especifica. Luego, se puede ver el todo como una sucesión de actividades de alto nivel sin necesidad de comprender los detalles.

Extensibilidad ya que se pueden incorporar y eliminar reglas, y alterar el orden de las reglas obteniendo un nuevo método de generación de álgebras.

A continuación presentamos las reglas que utilizamos para la construcción sistemática de expresiones algebraicas.

Existen cuatro conjuntos de reglas:

- un conjunto de reglas asociadas a la cláusula *select*,
- un conjunto de reglas asociadas a la cláusula *from*,
- un conjunto de reglas asociadas a la cláusula *where* referentes a la operación AND.
- un conjunto de reglas asociadas a la cláusula *where* referentes a la operación AND.

Las reglas asociados a la cláusula *select* son las últimas reglas que se aplican. Su objetivo es obtener la salida adecuada (dada por los tipos de salida) desde el conjunto de resultados. Generan operadores *ObjectProjection* o *AttributeProjection* según corresponda.

Las reglas asociadas a la cláusula *from* se aplican inmediatamente después del conjunto de reglas asociadas a la cláusula *where*, pero antes de las reglas asociadas a la cláusula *select*. Su objetivo es ligar al resultado aquellas variables que no han sido

utilizadas durante el procesamiento de la cláusula *where*. La forma de ligar dichas variables es utilizando la operación *CartesianProduct* entre el resultado y cada extensión de tipo no referenciada en el *where*. De esto se desprende que en el caso extremo de no tener *where* (es decir, un *where* nulo), se realiza el producto cartesiano entre todas las extensiones de tipos que aparecen en la cláusula *from*.

Existen dos conjuntos de reglas asociadas a la cláusula *where*:

Conjunto de reglas AND:

A este nivel solo contamos con descriptores de operaciones que están relacionadas con el operador lógico AND. Las reglas son:

> Regla 1: Unir todos los descriptores de operaciones de selección sobre la misma variable en un solo descriptor de selección sobre esa variable, y donde la condición resultante esta dada por las condiciones de todos los descriptores unidas con AND. Se repite la aplicación de esta regla hasta que ya no se pueda aplicar.
>
> Regla 2: Tomar dos descriptores cualesquiera y reemplazarlo por otro, que es el producto natural entre ambos. Se repite hasta que no se pueda aplicar.

La primer regla reduce el repositorio de descriptores optimizando la consulta. Reduce la cantidad de veces que debe recorrerse una extensión de tipos. Por ejemplo, si hay una selección de libros por título y otra por autor, cambia las dos por una sola que selecciona por ambas condiciones al mismo tiempo.

La segunda regla realiza el AND propiamente dicho. Supongamos dos expresiones booleanas A y B relacionadas con AND. El resultado esta formado por cada conjunto resultado que cumple la expresión A asociado a cada conjunto resultado que cumple la condición B (un producto cartesiano entre A y B).

Si ambas expresiones involucran las mismas variables, el valor de esas variables deben ser iguales en un conjunto resultado, esto debido a que en un momento dado una variable contiene un solo valor. De aquí surge que el AND entre dos expresiones se puede traducir como un producto natural (operador *NaturalProduct*).

La expresión que se genera no es la óptima, será responsabilidad del optimizador intentar llegar a la expresión óptima a partir de la expresión generada por el builder.

Conjunto de reglas OR:

Las reglas OR trabajan sobre descriptores que están asociados con el operador OR, y en este caso solo se aplica la siguiente regla de construcción:

> Regla 1: Tomar dos descriptores y reemplazarlos por uno que resulta de la combinación de estos dos. Se repite la operación hasta que no se pueda aplicar.
>
> El descriptor resultante se obtiene de la siguiente forma:
>
> Sean A=(t1,t2) y B=(t1,t3) dos expresiones y su estructura de tipos.
>
> a) Se realiza el producto cartesiano de A con cada tipo de B que no este incluido en A, es decir, el primer resultado seria $R1 = A \times \gamma(t3)$.

b) Se realiza la actividad análoga con B, y el segundo resultado esta formado por R2 = B X γ(t2).

c) Luego se deben reacomodar (mediante la proyección de objetos) las estructura de tipo para realizar finalmente la unión de los resultados obtenidos:

R1 U (t1,t2,t3[R2])

Esta única regla especifica la relación de OR entre dos operaciones. Si tenemos dos expresiones (A y B), el resultado de la consulta serán todos los conjuntos resultados que cumplen A, asociado a cada posible conjunto que tenga la estructura de B (puede o no cumplir B), y esto unido a todos los conjuntos resultados que cumplen B asociado a cada posible conjunto que tenga la estructura de A (que puede o no cumplir A). Esto es debido a que alcanza con que se cumpla una de las condiciones para ser un resultado válido de la consulta.
Traducido a términos algebraicos, se puede representar "todos los posibles conjuntos que con la estructura de B" como el producto cartesiano entre todas las extensiones tipos de B, esto asociado a "todos los conjuntos resultados que cumplen con A" seria el producto cartesiano entre A y lo obtenido por el producto cartesiano entre todas las extensiones de tipos de B.
Cada variable en un momento dado posee un solo nodo asociado, por lo tanto no hay que generar repeticiones, es decir, no puede aparecer una tupla con dos columnas del tipo Autor que posean distintos autores. De allí surge la restricción de hacer el producto cartesiano de A solo con los tipos de B que no estén en A. La inversa es análoga.

Antes de aplicar las reglas del *from*, hay que pedirle al árbol de descriptores que marque como usadas las variables que está referenciando.

Resultado:

Luego de la aplicación de todas las reglas antes expuestas obtenemos un árbol de descriptores de operaciones.
Dado este árbol se puede obtener directamente el álgebra final que será pasada al optimizador.

En el capítulo siguiente, veremos como fueron implementados los componentes previamente descriptos, como así también los distintos operadores algebraicos definidos en este capítulo.

Capítulo 7.
Implementación de OOHQL.

7.1 Introducción.

En esta sección hablaremos de las consideraciones tenidas en cuenta antes y durante la implementación del lenguaje OOHQL, es decir, del motor de resolución de consultas que permite ejecutar consultas expresadas en OOHQL sobre una aplicación hipermedia orientada a objetos, e instanciada con el OO-Navigator Framework. Este motor es lo que denominamos el QSC (*Query Solving Component*).

Si bien ejemplificaremos la etapa de implementación aquí descripta, con extractos del código fuente desarrollado, no se intenta en esta sección, comentar los detalles particulares del proceso de codificación, por considerarlos irrelevantes de acuerdo al objetivo de esta tesis, planteado anteriormente. Sí trataremos de dar una visión general sobre las ventajas e inconvenientes del lenguaje elegido, ejemplificando aquellas consideraciones con algunos diseños específicos.

Por último presentaremos la herramienta gráfica desarrollada para la escritura de consultas y visualización de los respectivos resultados.

Este capítulo esta organizado de la siguiente forma: en la sección 7.2 se presentan las características de la plataforma utilizada para el desarrollo. En la sección 7.3 se explica la arquitectura del sistema. En la sección 7.4 se detalla la implementación del álgebra propuesta en el capítulo anterior. En la sección 7.5 se explica la implementación de los mecanismos de traducción del string OOHQL a una representación algebraica equivalente. Finalmente, en la sección 7.6 se presentan las herramientas visuales del QSC.

7.2 Consideraciones sobre el lenguaje utilizado y el ambiente de desarrollo.

El lenguaje de implementación elegido para este trabajo es Smalltalk-80 [Goldberg83]. El mismo es un lenguaje OO puro, es decir, que respeta consistentemente el paradigma de orientación a objetos, provee herencia simple, polimorfismo, binding-dinámico y es no-tipado. En ambiente de desarrollo utilizado durante el proceso de implementación y testeo del sistema, fue VisualWorksTM 3.0.

El ambiente provisto por VisualWorks brinda: categorización de clases y de métodos dentro de una clase, manejo de proyectos para trabajo colaborativo, manejo de cambios generales y por proyecto, debugging especializado, manejo de excepciones, una herramienta de diseño de interfaces en forma visual llamada "canvas", browsers de clases, de jerarquías, de categorías de clases y métodos, entre las características más salientes [PP94].

Entre las ventajas obtenidas con la utilización del Smalltalk de VisualWorks podemos enumerar:

Implementación directa del diseño: ningún artificio del lenguaje de implementación es necesario cuando se utiliza Smalltalk. Los componentes modelados en la etapa de diseño aparecen casi idénticos, aunque más detallados, en la codificación,

por tratarse de un lenguaje de muy alto nivel. Por esto generalmente se dice que la implementación en Smalltalk es casi la transcripción del diseño.

Limpieza del código: no son necesarias construcciones especiales como ocurre con lenguajes híbridos, ni es necesaria la destrucción explícita de instancias gracias al recolector de basura provisto por el ambiente.

Múltiples plataformas: la misma imagen construida en una plataforma entre Windows, Macintosh, OS/2 o UNIX, puede ser usada en cualquiera de las otras sin ningún cambio necesario. Esta característica hace que el QSC sea completamente portable.

Mecanismo de dependencias: este mecanismo provisto por el lenguaje permite la correcta implementación de "observadores" u objetos que se hacen dependientes de los cambios producidos sobre objetos observados o "modelos", sin necesidad de que estos últimos tengan que avisar a sus dependientes en forma explícita. Cualquier objeto puede tener dependientes, aunque las instancias de las subclases de Model están mejor preparadas para mantener dependientes en forma eficiente. Cada vez que un modelo cambia en alguna forma que puede ser de interés para al menos un dependiente, debe realizar un broadcast notificando de su cambio a todos sus dependientes, que a su vez decidirán si están interesados en el cambio. Este mecanismo de notificación entre un modelo y sus dependientes se denomina "changed/update" [Howard95]. Existen dos tipos de modelos dependiendo de su granularidad: modelos de la aplicación o "application models", y modelos de valor o aspectos, llamados "value models". Un "application model" es responsable de manejar las partes de una ventana, y es construido para una aplicación específica. Por su lado, un "value model" contiene un único aspecto de información que constituye su valor. Este concepto es tan importante en VisualWorks que todos los widgets (partes atómicas de una ventana) usan value models como modelos. Existen varios tipos de value models, cada uno representado por una subclase de ValueModel, como ValueHolder, AspectAdaptor y PluggableAdaptor. El concepto de "value model" se potencia además con el de "dependency transformer" para optimizar el mecanismo de dependencias [Woolf94].

Este mecanismo permitió implementar las herramientas visuales del QSC, que permiten realizar consultas y navegar los resultados, en forma totalmente independiente del modelo de objetos subyacente. De esta manera, podríamos en el futuro implementar otras interfaces sobre el mismo modelo, sin necesidad de alterar el QSC.

Arquitectura en capas independientes: una arquitectura en capas es aquella en la que el diseño de cada capa es independiente, e introduce cierto protocolo que restringe la interface de la misma [Campbell91]. En Smalltalk, esto es totalmente factible de ser implementado, en parte gracias al mecanismo de dependencias visto anteriormente y en parte por la explícita separación que se hace entre los objetos del dominio y su interface. Esta separación es fundamental para el desarrollo de una arquitectura flexible, mantenible, reusable y modificable. Esta característica posibilitó y facilitó en gran medida la implementación del QSC como una capa totalmente independiente de la aplicación hipermedial instanciada con el OO-Navigator.

7.3 Arquitectura del sistema.

Desde el punto de vista de la implementación de un sistema hipermedial que permita realizar consultas, han sido definidas tres capas: 1) un ambiente de desarrollo orientado a objetos donde se desarrolla la aplicación de objetos, 2) el OO-Navigator que agrega funcionalidad hipermedial a la aplicación anterior, y 3) el motor de consultas o QSC. La interacción entre estas capas fue descripta anteriormente en el Capítulo 5. En lo que sigue de este capítulo, nos concentraremos en la arquitectura del QSC y su implementación.

El QSC tiene como responsabilidad recuperar la información deseada y permitir al usuario final navegar el resultado de su consulta. Una consulta está formada por su especificación (el string OOHQL) y la indicación de la estrategia de navegación que se desea utilizar para acceder al resultado de la misma.

El QSC está compuesto por cuatro tipos de objetos diferentes:

- Un *HMQuery*: que posee diversas responsabilidades. Es quien representa a la consulta como un objeto, almacena el resultado de la consulta y lleva a cabo el proceso de resolución de la misma, colaborando con los tres componentes que siguen abajo.
- Un *HMQueryParser*: que transforma la expresión de la consulta (el string OOHQL) en una expresión algebraica, es decir, una expresión OOHA bien formada, colaborando con un builder.
- Un *HMQueryOptimizer*: que optimiza la expresión algebraica que genera el componente anterior de acuerdo a un conjunto de reglas de optimización particulares.
- Un *HMQueryEngine*: que dispara y supervisa la ejecución de la consulta, generando posteriormente el resultado.

La Figura 7.1 muestra la interacción de estos componentes.

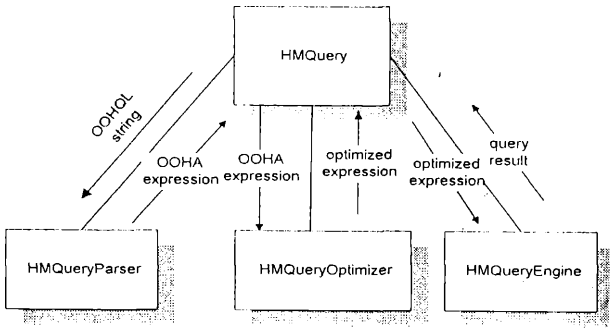


Figura 7.1 – Interacción de los componentes que conforman el HMQuery.

En la Figura 1 mostramos los componentes más importantes del sistema y su interacción. A continuación, vamos a entrar más en detalle en la composición del *HMQuery*, el componente más importante de los cuatro descriptos anteriormente, ya que es quien representa la consulta en si misma.

La Figura 7.2 presenta el diagrama OMT [Rumbaugh91] que especifica las relaciones de conocimiento y composición del *HMQuery* e introduce algunos objetos nuevos que fueron usados para la implementación del QSC, y que detallaremos a continuación.

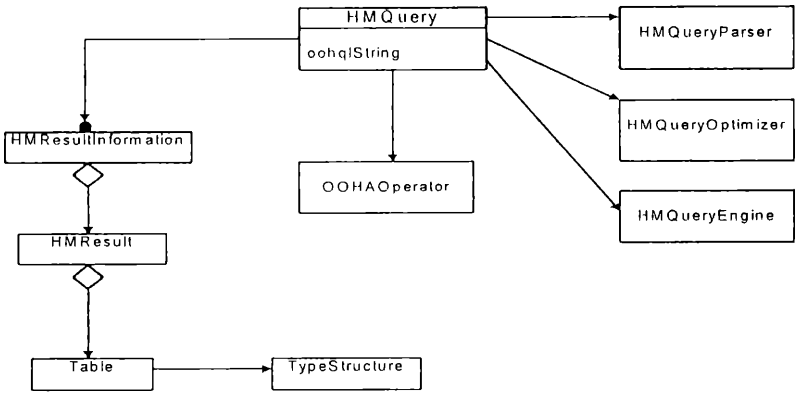


Figura 7.2 – El *HMQuery* en detalle.

Como describimos anteriormente, el *HMQuery* conoce al *HMQueryParser, HMQueryOptimizer* y al *HMQueryEngine.*

Además, el *HMQuery* posee las siguientes componentes:

- El OOHQLString que representa la expresión, en OOHQL, de la consulta. Se mantiene esta referencia con el objetivo de permitirle al usuario la modificación de una consulta sin tener que reescribir toda la consulta nuevamente.
- El álgebra retornada por el *HMQueryOptimizer*. Esta representación algebraica es totalmente independiente de la aplicación hipermedial sobre la cual se ejecute la consulta. El álgebra será utilizada por el *HMQueryEngine* para ejecutar la consulta. Se mantiene esta referencia con el objetivo de no tener que resolver toda la consulta cuando se desee realizar un "update" de la misma.
- La colección de *ResultInformation*, que fue pensada para proveer la facilidad de que una misma consulta pueda tener varios resultados asociados, dependiendo de la hipermedia, fecha y hora en que se halla ejecutado.

En el momento de ejecutar una consulta, el resultado es guardado junto a una fecha y una hora determinada. Para acceder a dicho resultado el usuario debe, además de seleccionarlo, especificar con que estrategia de navegación va a accederlo. Solo se puede acceder a un resultado de una consulta con una estrategia de navegación asociada.

A continuación, vamos a definir las responsabilidades de los demás objetos que aparecen en la Figura 7.2.

Table:

Una tabla es la forma general de mantener las instancias agrupadas en forma de conjunto, y posee una estructura que determina qué cosas almacena y de qué forma. En particular, cada operación del álgebra antes definida (salvo la *Identity* y *AttributeProjection*) toma como argumento una tabla y retorna una tabla como resultado.

Este objeto es solo manipulado dentro del sistema de consultas y no llega nunca al usuario. En realidad, ni siquiera es manipulado fuera del *HMQueryEngine* que es quien ejecuta el álgebra y toma la tabla como base para generar el resultado según la especificación hecha por el usuario.

Type Structure:

Un *TypeStructure* o simplemente estructura de tipo, representa la estructura, en cuanto a tipo de nodos y orden de los mismos, dentro de una tabla. De esta forma representamos la estructura de la tabla para poder saber exactamente como manipularla internamente cuando se resuelven los operadores del álgebra anteriormente presentada. La estructura de tipos de una tabla no es visible para quien realiza una consulta.

HMResult:

Este objeto representa el resultado de una consulta y está formado por:

- una tabla que contiene el resultado final de la evaluación de la consulta,
- y el objeto de protocolo hipermedial (hipermedia u otro *HMResult*) sobre el que se ejecutó la consulta.

Este objeto sirve como entrada a cualquier otra consulta, siempre y cuando la tabla que contenga, almacene nodos y no atributos, ya que no es posible hacer una consulta tomando como hipermedia una proyección de atributos de nodos.

Las razones de separar la abstracción del resultado de una consulta, de la consulta en si misma, son:

- Nos permite tener varios resultados asociados a una misma consulta. Un usuario puede querer evaluar una misma consulta sobre diferentes hipermedias, o comparar resultados obtenidos de una consulta sobre una misma hipermedia en diferentes momentos.
- Mientras que el resultado está siendo navegado o utilizado como fuente de información para otra consulta, no es necesario que mantenga referencias al parser, al optimizador, al álgebra, etc.

HMResultInformation:

Este objeto surge a partir de la necesidad de permitir una diferenciación explícita de los distintos resultados de una misma consulta. Representa datos sobre el resultado pero que no forman parte del mismo.

Sus componentes son:

- _fecha: en la que fue ejecutada la consulta.
- _hora: a la que fue ejecutada la consulta.
- _nombre: identificador que el usuario le asigna a una consulta determinada y que lo ayuda a comprender su contenido.
- _hmResult: referencia al resultado propiamente dicho.

El "update" es una opción permitida por la consulta solo cuando tiene seleccionado alguno de estos resultados posibles.

7.4 Implementación de OOHA.

En el Capítulo 6 definimos un álgebra que nos permite expresar formalmente una consulta OOHQL. El proceso de ejecución de una consulta, descripto en la Sección 6.4, especifica que la consulta OOHQL debe ser traducida a una expresión OOHA, ésta expresión debe ser luego optimizada y por último ejecutada sobre una aplicación hipermedial.

Con tal motivo, en esta sección vamos a definir la forma en que implementamos los diversos operadores OOHA que permiten expresar consultas OOHQL.

Las Figuras 7.3.a, 7.3.b y 7.3.c, definen la jerarquía de clases que implementa los operadores OOHA. Se ha dividido la jerarquía en tres figuras para favorecer su comprensión y claridad.

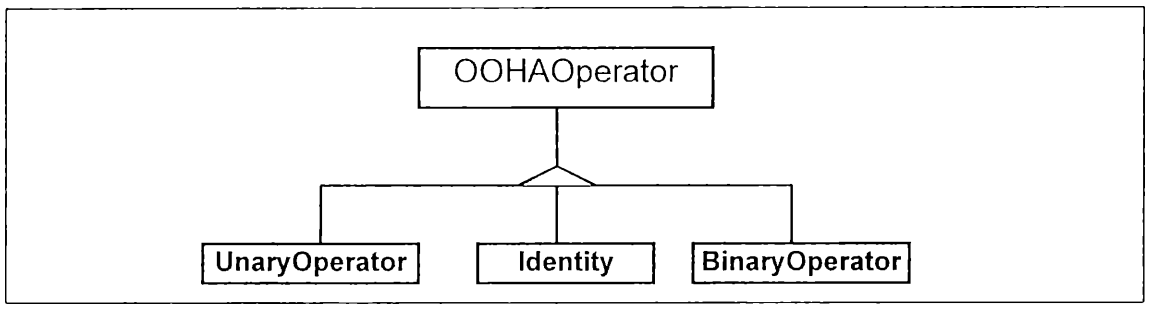


Figura 7.3.a – Subclases de OOHAOperator.

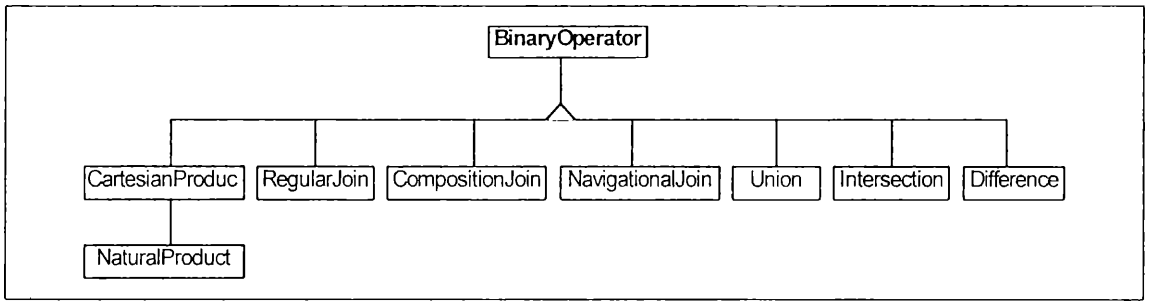


Figura 7.3.b – Subclases de BinaryOperator.

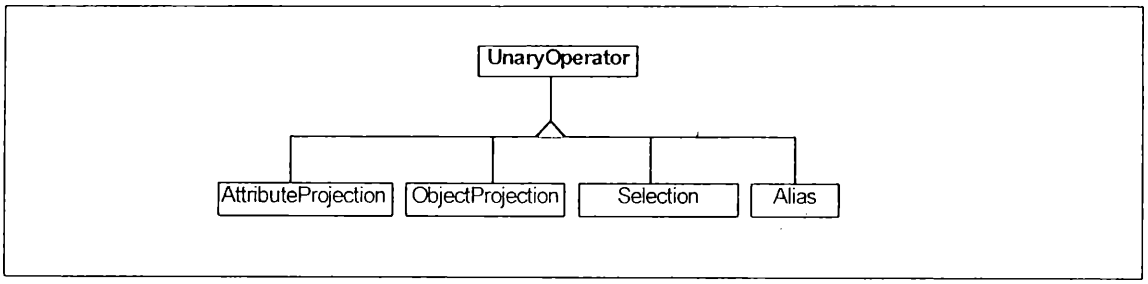


Figura 7.3.c – Subclases de UnaryOperator.

Todos los operadores OOHA son subclase de *OOHAOperator*. Esta clase define el protocolo de mensajes comunes a todos los operadores. El mensaje más importante que todo operador OOHA debe responder es executeOn: aHypermedia. Este mensaje no está implementado en OOHAOperator, sino que debe ser redefinido por sus subclases.

En particular, el mensaje executeOn: está redefinido en *UnaryOperator* y en *BinaryOperator*, como puede verse a continuación en los siguientes extractos de código fuente:

- **UnaryOperator>>executeOn: aHypermedia**

```
"Execute the algebra, return the table result"

| resultTable childTable myStructure |
childTable := self child executeOn: aHypermedia.
myStructure := self resultTypeStructure: childTable typeStructure.
resultTable := Table typeStructure: myStructure.
^self
        evaluateWith: childTable
        in: resultTable
        on: aHypermedia
```

- **BinaryOperator>>executeOn: aHypermedia**

```
"Executes the receiver on aHypermedia and answers a table containing the result."

| leftTable rightTable myTypeStructure resultTable |
leftTable := self leftChild executeOn: aHypermedia.
rightTable := self rightChild executeOn: aHypermedia.
myTypeStructure := self makeTypeStructureWith: leftTable typeStructure and:
rightTable typeStructure.
resultTable := Table typeStructure: myTypeStructure.
^self
        join: leftTable
        with: rightTable
        in: resultTable
        on: aHypermedia
```

Como puede observarse en el código fuente, el mensaje executeOn: es un Template Method [Gamma95]. Los mensajes join:with:in:on: y evaluateWith:in:on: son los correspondientes hook methods y deben ser implementados tanto en las subclases de *UnaryOperator* como de *BinaryOperator*. Por ejemplo, la clase *Selection*, subclase de *UnaryOperator*, que implementa el operador de selección de OOHA, implementa el mensaje evaluateWith:in:on: de la siguiente manera:

- **Selection>>evaluateWith: childTable in: resultTable on: aHypermedia**

```
"Execute the receiver on childTable, and stores the result in resultTable. Returns
resultTable"

childTable doTuples: [:tuple | (self block value: tuple)
                ifTrue: [resultTable addTuple: tuple]].
^resultTable
```

Por otro lado, la clase *NavigationalJoin*, subclase de *BinaryOperator*, que implementa el operador de join navegacional de OOHA, implementa el mensaje join:with:in:on: de la siguiente manera:

- **NavigationalJoin>>join: leftTable with: rightTable in: resultTable on: aHypermedia**

```
"Private - Returns the resultTalbe with the NavigationalJoin betwen leftTable and
rightTable."

| linkClasses |

linkClasses := self linkClassesFromTypesIn: aHypermedia.
leftTable doTuples:[ :x1 |
                    rightTable doTuples:[ :x2 |
                                        (self verifyPathFrom: (x1 at: self originType) to: (x2 at:
self destinyType) across: linkClasses )
                                                ifTrue: [resultTable addTuple: (x1 , x2)]
                                ]].
^resultTable
```

Finalmente, es interesante detenerse en como implementa la clase *Identity*, subclase directa de *OOHAOperator*, el mensaje **executeOn:**, ya que es en esta clase donde se interactúa con el OO-Navigator para recuperar las instancias de nodos que serán utilizadas en la consulta.

- **Identity>>executeOn: aHypermedia**

```
"Evaluates the receiver on aHypermedia. Returns a Table as result."

| nodeClass typeStructure table tupla |
nodeClass := aHypermedia nodeClasses detect: [:nc | nc name asSymbol = self child]
                        ifNone: [^self error: ' Invalid type for this hypermedia: '
                        , self child asString].
typeStructure := TypeStructure structure: (Array with: (Type type: self child)).
table := Table typeStructure: typeStructure.
nodeClass nodes
        do:
                [:n |
                tupla := Tuple typeStructure: typeStructure.
                tupla add: n.
                table addTuple: tupla].
^table
```

7.5. Implementación del proceso de traducción algebraica.

En esta sección, detallaremos la forma en que fueron implementados el Parser, el Builder y las reglas de transformación algebraica presentadas en el capítulo anterior.

7.5.1 Implementación del Parser.

El intérprete sintáctico del lenguaje fue implementado utilizando una herramienta de generación automática de parsers para Smalltalk, denominada T-Gen [T-Gen]. El T-Gen permite especificar la gramática del lenguaje que se quiere interpretar, y genera en forma automática, un parser que toma como entrada un string y verifica que el mismo esté dentro del conjunto de strings generados por dicha gramática, es decir, dentro del lenguaje de la gramática. La gramática desarrollada durante esta tesis para interpretar strings OOHQL, es decir la gramática del lenguaje OOHQL, puede encontrarse en el Apéndice B de la presente tesis.

Como resultado de la utilización del T-Gen, se genera una clase (a partir de la cual se instancia el parser) que denominamos *OOHADescriptorParser*.
El parser no solo verifica que el string OOHQL esté correctamente escrito, sino que también arroja un resultado. Para cada operación referida en la consulta OOHQL, se genera un descriptor, de allí el nombre de la clase generada por el T-Gen. Los descriptores ya fueron presentados con anterioridad en el capítulo anterior. De todas formas, su función es representar cada una de las operaciones necesarias para resolver la consulta. Son en realidad una representación intermedia de la consulta que se encuentra entre el string OOHQL y la correspondiente expresión OOHA. Precisamente, la expresión algebraica se obtiene a partir del procesamiento de los descriptores.
Para generar los descriptores, el parser colabora (enviándole ciertos mensajes) con otro objeto que describiremos a continuación, el builder.

7.5.2 Implementación del Builder.

El generador de descriptores o builder, también fue implementado utilizando el T-Gen. T-Gen permite no solo especificar la gramática del lenguaje, sino también determinar que mensajes se le enviarán al builder para generar la salida deseada, que en este caso es el conjunto de descriptores que representan las operaciones necesarias para satisfacer la consulta. Estos mensajes son enviados desde el parser hacia el builder cada vez que se identifica una construcción particular de la gramática, la cual debe tener asociada el selector del mensaje a enviar. El builder es instancia de la clase *OOHADescriptorBuilder*.
Una vez que el builder ha generado la representación equivalente del string OOHQL, es decir, el conjunto de descriptores que representan a la consulta, el mismo builder puede tomar como entrada estos descriptores para generar finalmente la expresión algebraica. Para esto aplica las reglas descriptas en el capítulo anterior, sobre el conjunto de descriptores generado anteriormente. Este conjunto de descriptores generado por el builder tiene la estructura de un árbol, es decir, es exactamente la misma estructura de anidación de operaciones que tendrá la expresión OOHA que será ejecutada. Cada descriptor entiende el mensaje *buildAlgebra* al cual responde retornando una instancia de la subclase de *OOHAOperator* que corresponda. Por ejemplo, si a un *SelectionDescriptor* se le envia el mensaje *buildAlgebra*, retorna una instancia de *Selection* configurada según la información del descriptor.
A continuación veremos como fueron implementadas las reglas de transformación de descriptores que generan la expresión OOHA.

7.5.3 Implementación de las reglas de transformación.

Las reglas de transformación algebraica fueron implementadas siguiendo el pattern Strategy [Gamma95]. Cada regla fue modelada como un objeto que encapsula el algoritmo de transformación correspondiente.
Una vez que el parser ha verificado la correctitud sintáctica del string OOHQL y el builder ha generado los descriptores de cada operación, se aplican las reglas descriptas en el capítulo anterior para generar el árbol de descriptores que finalmente se transformará en la expressión OOHA ejecutable. Además de los descriptores, en el capítulo anterior hicimos referencia a los repositorios. Los repositorios son instancia de alguna de las subclases de *Repository*, es decir, *AndRepository* u *OrRepository*. Cada una de estas subclases genera un repositorio configurado con las reglas de

transformación correspondientes. Los repositorios y los descriptores comparten un mismo tipo, es decir, ambos entienden un conjunto de mensajes que les permiten generar una expresión OOHA. En particular, ambos (repositorios y descriptores) entienden el mensaje *buildAlgebra*. Como explicamos anteriormente, un descriptor responde a este mensaje retornando la instancia de *OOHAOperator* correspondiente. Un descriptor, finalmente también retorna una instancia de *OOHAOperator*, pero para esto evalúa las reglas con las que fue configurado sobre los descriptores que contiene. De esta forma, trabajamos polimórficamente con los repositorios y los descriptores.
Una vez que el árbol de descriptores ha sido generado, basta con enviarle el mensaje *buildAlgebra* a la primera operación OOHA del árbol, es decir, a la última operación que se ejecutará. Esto provoca que los hijos de esa operación se conviertan en operaciones OOHA, y luego el receptor original se convierte en una operación OOHA que tiene como hijos a las operaciones que son hijos directos.

7.6. Herramientas visuales del QSC.

Finalmente, en esta última sección, vamos a describir las herramientas visuales provistas con la implementación del QSC, que permiten expresar consultas en OOHQL, ejecutarlas, y visualizar los resultados. Como describimos anteriormente, estas interfaces de usuario fueron desarrolladas utilizando la implementación del framework MVC (*Model View Controller*) de VisualWorks. De esta forma, las interfaces del QSC son totalmente independientes del modelo de objetos utilizado para implementar el QSC, que fue descripto en este capítulo. Es decir, las interfaces propuestas a continuación son solo una aproximación a la visualización de las consultas OOHQL, propuesta por nosotros. Eventualmente, estas interfaces podrían ser alteradas, podrían agregarse nuevas interfaces e incluso eliminarlas, sin que el modelo del QSC tenga que ser alterado.

La Figura 7.4 muestra la interface de edición de consultas del QSC.

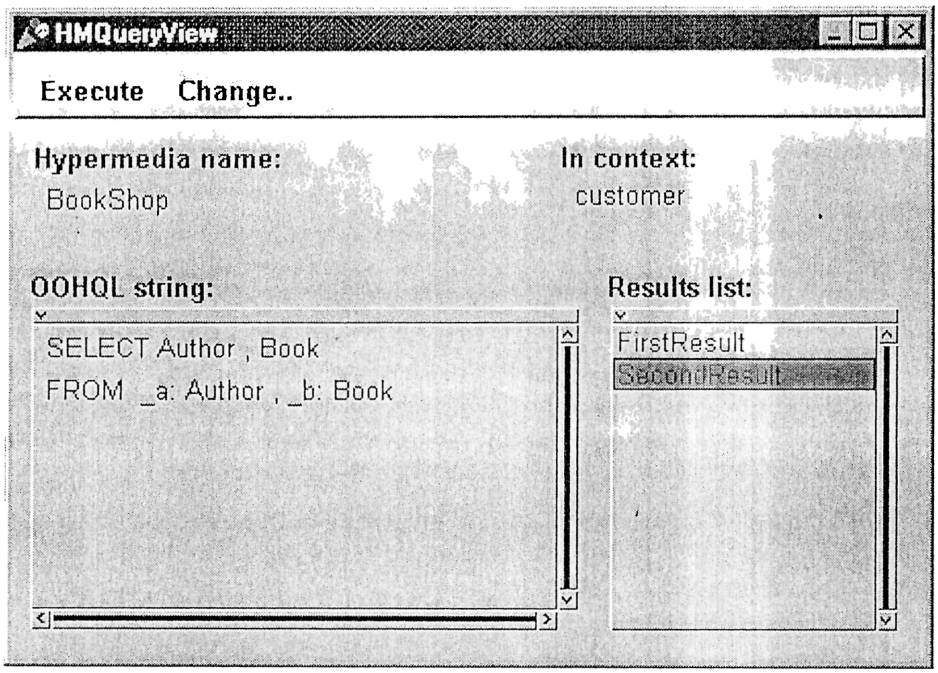


Figura 7.4 - El HMQueryView.

En esta figura podemos observar la interface principal de edición de consultas. Esta interface es instancia de la clase *HMQueryView* y permite seleccionar el contexto en el cual se ejecutará la consulta. Cabe recordar que una aplicación hipermedia puede tener diversos contextos de usuario definidos. De esta forma, el usuario debe tener presente que solo podrá consultar información habilitada en su contexto de usuario. Es decir, si un "customer" no permite acceder, por ejemplo, a los autores de un libro, la consulta que se expresa en la Figura 7.4 no podría ejecutarse.

Además, la interface también permite seleccionar la aplicación hipermedial sobre la cual se va a ejecutar la consulta. Recordemos que esta aplicación puede ser un hipermedia o un resultado previo de otra consulta.

Por último, la interface permite crear y editar el string OOHQL que da origen a la consulta, ejecutarlo y acceder a su resultado. Cada consulta puede tener más de un resultado asociado, y dichos resultados se listan en el pane llamado "Result list:".

En la figura 7.5 se muestra el diálogo que nos consulta sobre que estructura de acceso vamos a utilizar para observar el resultado.

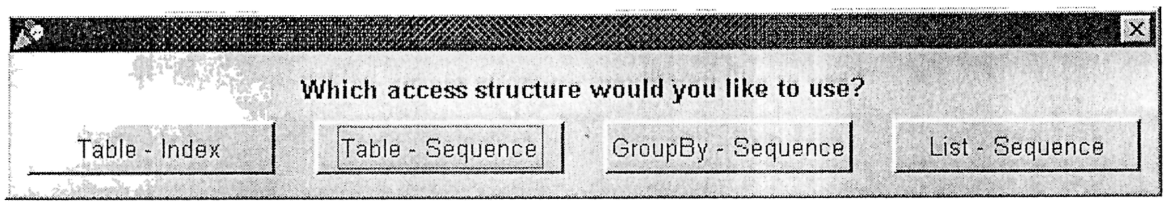


Figura 7.5. Diálogo de consulta de estructuras de acceso.

En la figura 7.6 veremos la aplicación relacionada al acceso indexado.

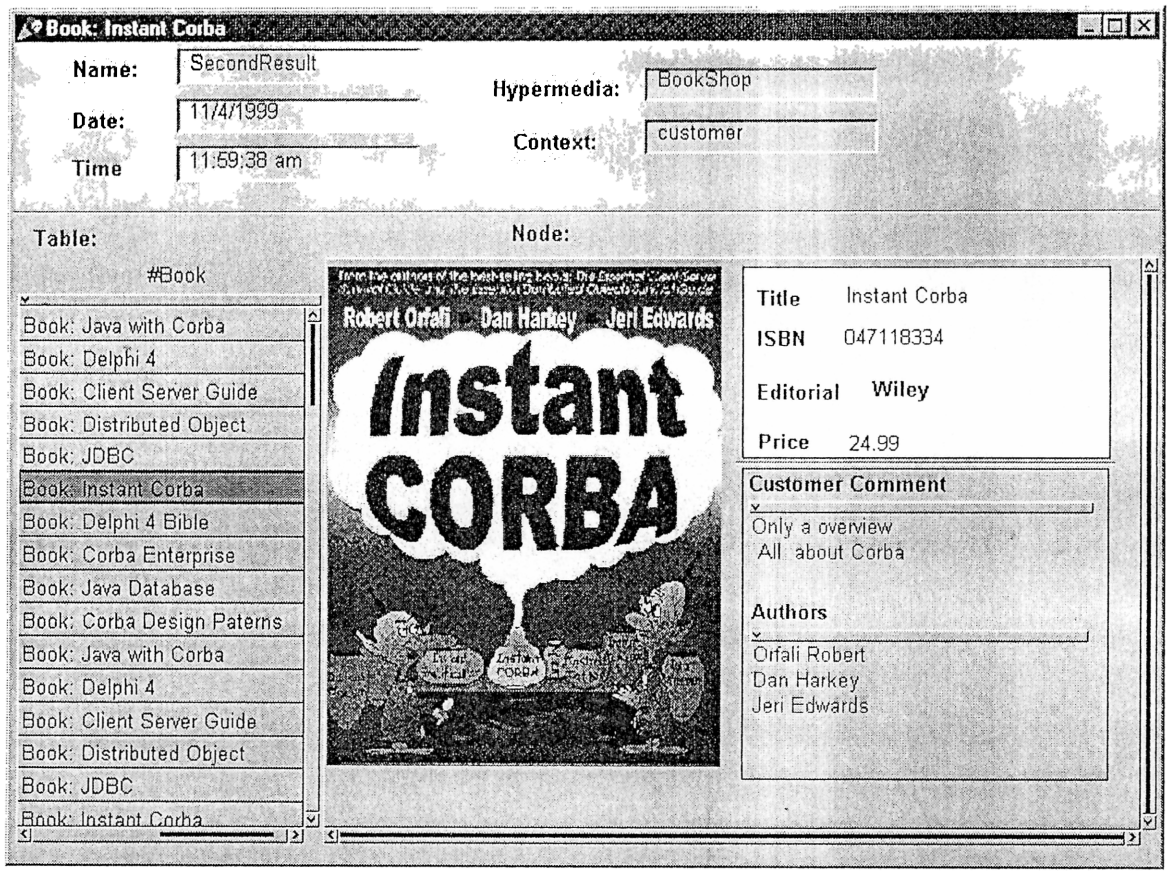


Figura 7.6 – Acceso indexado a un resultado en forma de tabla.

Esta aplicación implementa la forma de acceso indexada al resultado de la consulta. El usuario puede seleccionar cualquier posición de la tabla y el nodo correspondiente se activa en el pane de la derecha llamado "Node". Cabe destacar que cuando se activa un anchor en el nodo que está siendo visualizado, se puede salir del resultado de la consulta. Por ejemplo, si siguiéramos el link a partir del anchor "Orfali Robert" navegaríamos fuera del resultado de la consulta original, aunque este autor esté

incluido en la tabla que se muestra en el pane de la derecha. Por esto al activar este link, se abrirá otra ventana con el nodo destino.

En la figura 7.7 se puede observar una aplicación que permite un acceso secuencial por navegación al resultado de una consulta en forma de tabla (bidimencional). Los accesos secuenciales tienen un link al anterior y siguiente conjunto resultado (verticalmente a la izquierda de la ventana), y a su vez dentro de cada resultado en particular se puede ir al anterior y al siguiente nodo (horizontalmente ubicados por debajo del nodo).

La Figuras 7.8 y 7.9 se muestran también herramientas secuenciales. En la aplicación 7.8 existe un único conjunto resultado que resulta de anexar todos los nodos que otorgo la consulta como un único resultado y en la aplicación 7.9 se atrapa el concepto de groupBy agrupando resultados según el tipo de nodo indicado en la parte inferior izquierda de la ventana. Ej: Consultamos Libro y Autor, al agrupar por Libro nos da un conjunto resultado formado por un libro junto a todos sus autores.

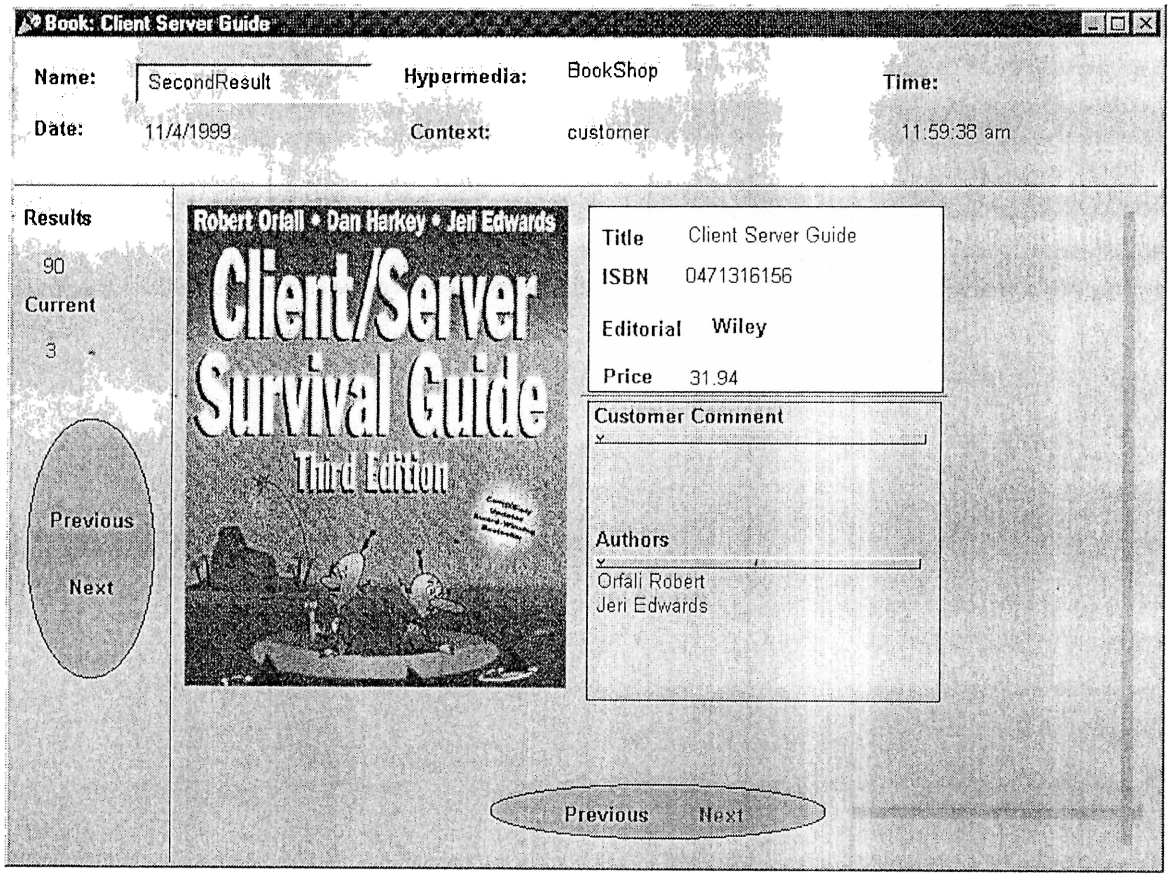


Figura 7.7. Acceso Secuencial en forma de tabla.

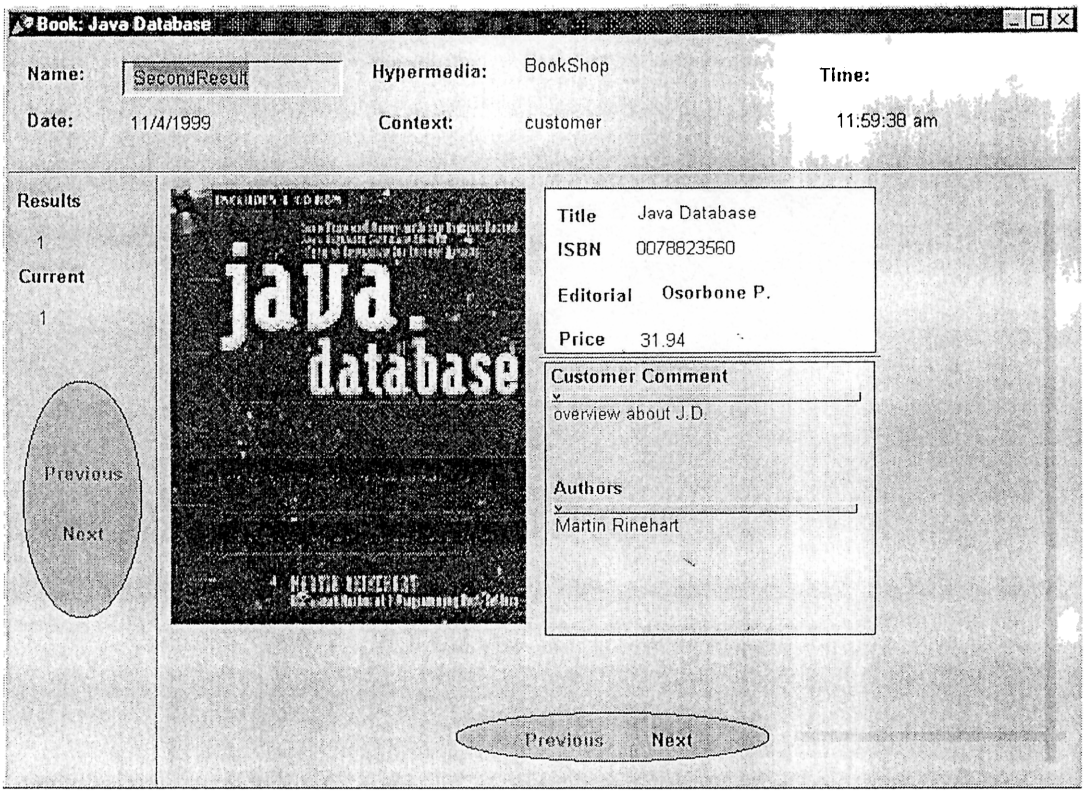


Figura 7.8. Acceso Secuencial en forma de lista.

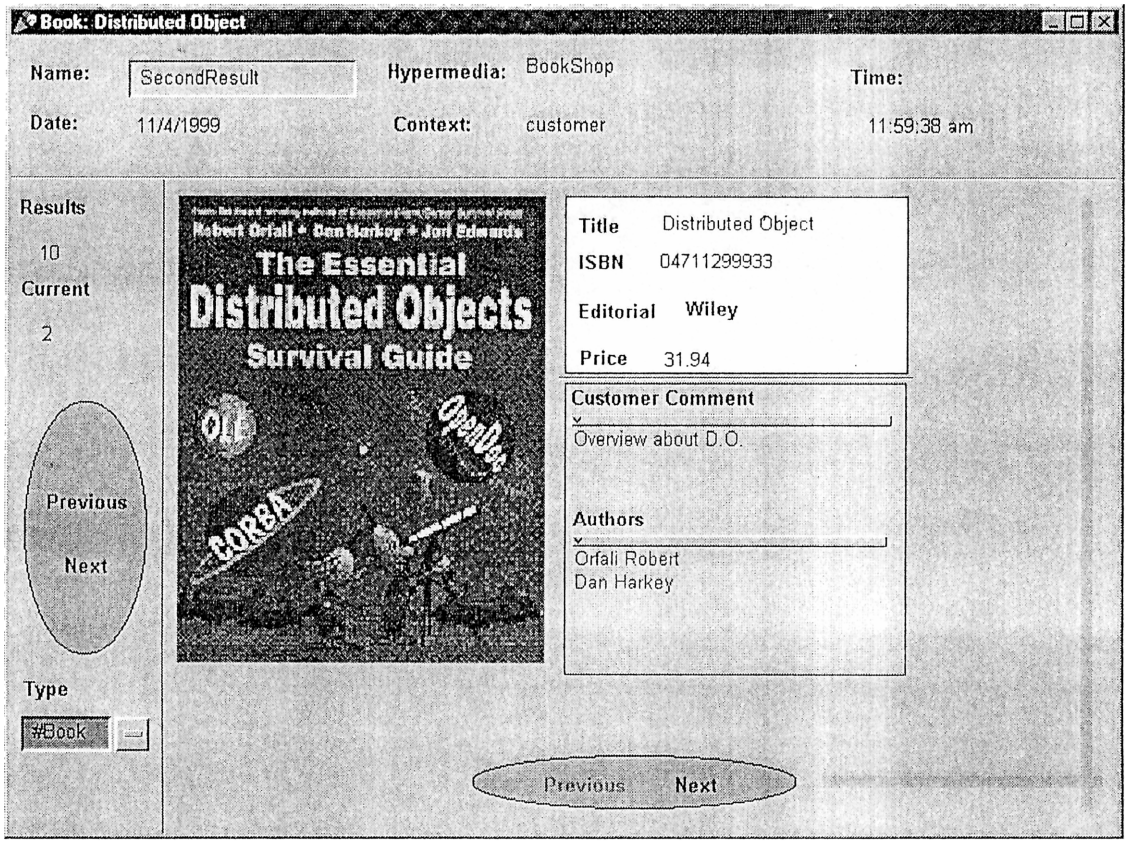


Figura 7.9. Acceso Secuencial agrupado por tipo de nodo.

Finalmente, la figura 7.10 muestra una consulta que arroja como resultado una proyección de atributos de nodos. Este tipo de consultas tienen una sola forma de ser visualizadas, mediante una interface particular que se muestra en la figura 7.11.

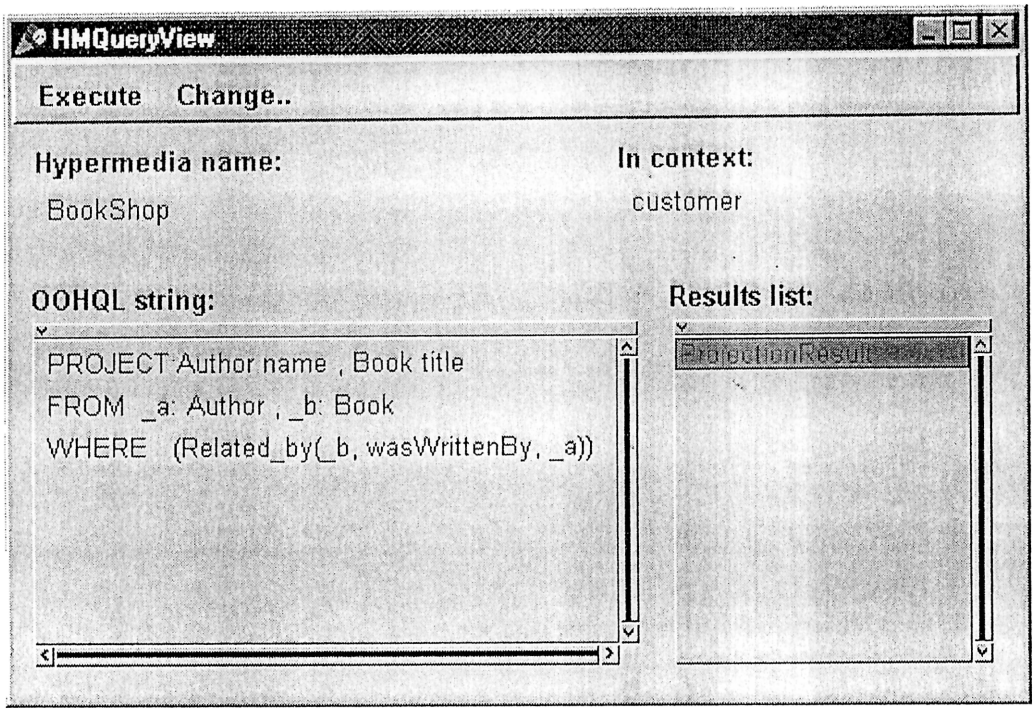


Figura 7.10 - Una consulta con proyección de atributos.

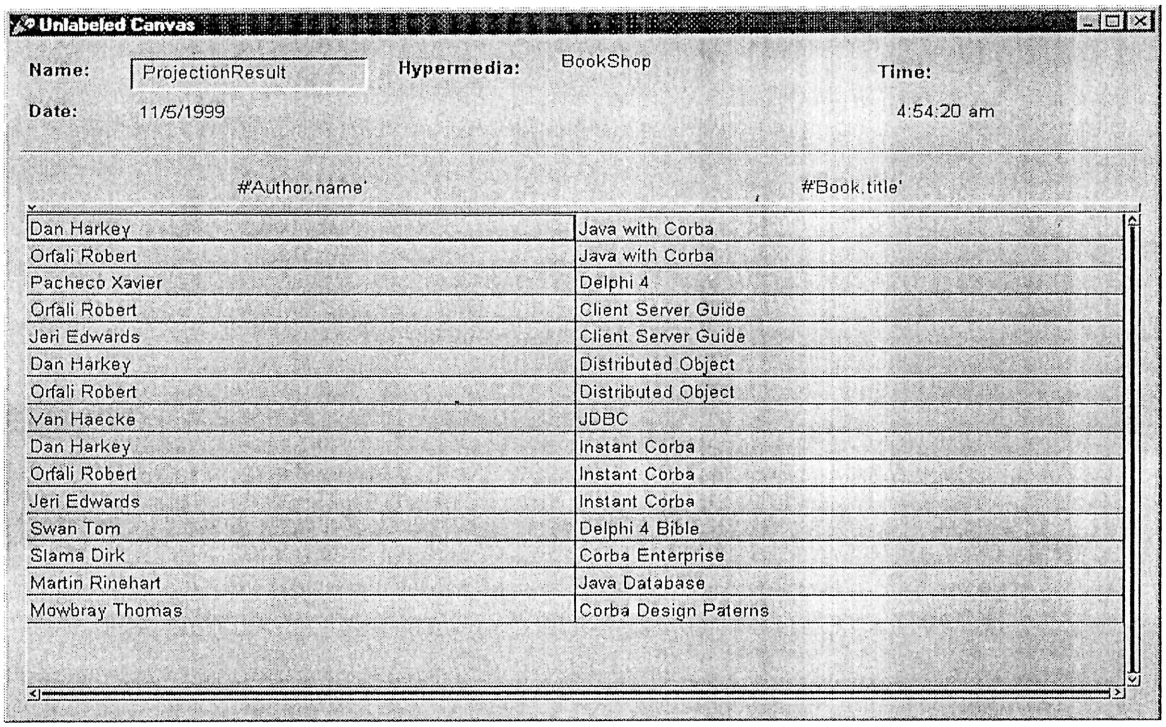


Figura 7.11 - Visualización de un resultado con proyección de atributos.

Capítulo 8.
Conclusiones y Trabajo Futuro.

Por último, este capítulo tiene como objetivo comentar las conclusiones a las que hemos arribado durante el desarrollo de la presente tesis, así como también establecer los lineamientos del trabajo futuro a realizar sobre el tema.

8.1 Conclusiones.

Nuestra tesis abarca la definición e implementación de un lenguaje de consultas para aplicaciones hipermediales orientadas a objetos. Este lenguaje, denominado OOHQL, ha sido definido sobre la base de un modelo de datos que se definió en el Capítulo 4. La definición y posterior implementación del lenguaje, nos ha permitido establecer que OOHQL es un lenguaje que permite la fácil y rápida recuperación de información de sistemas hipermediales orientados a objetos, favoreciendo la orientación de los usuarios dentro del espacio de navegación hipermedial, y aportando una herramienta para la solucionar la pérdida de los usuarios en aplicaciones hipermediales descripto en el Capítulo 1.

Aunque el lenguaje de consultas fue diseñado originalmente para trabajar con redes hipermediales convencionales, tales como enciclopedias interactivas, su implementación como capa superior al OO-Navigator Framework, ha probado ser particularmente satisfactoria. Hemos presentado algunos ejemplos sobre una empresa de venta de libros, que demuestran como el lenguaje puede ser también utilizado para consultar aplicaciones hipermediales que se basan en una aplicación de objetos subyacente sin características o funcionalidad hipermedial.

Por otra parte, debemos destacar la importancia de que el usuario del lenguaje debe conocer a priori el esquema navegacional de la aplicación a consultar, para poder realizar dichas consultas. Como comentamos anteriormente, esto es análogo al hecho de que para utilizar SQL se debe conocer el diagrama de entidad-relación de la base de datos relacional a consultar.

Hemos implementado el QSC con una arquitectura orientada a objetos que desacopla los procesos de parseo, optimización y procesamiento algebraico de la consulta, lo que nos permite un mantenimiento más sencillo y rápido de dicho componente. En este punto, debemos señalar que el álgebra que hemos definido nos permite especificar algoritmos complejos de resolución de consultas, mediante la composición de algoritmos más sencillos como son los operadores OOHA por separado. Además, esta especificación algebraica nos proporcionó una marco formal sobre el cual podemos realizar estudios de optimización de consultas sobre sistemas hipermediales.

Desde nuestro punto de vista, la definición de esta nueva álgebra, como extensión a las álgebras relacionales de Codd, son el mayor aporte realizado por esta tesis, ya que brinda una herramienta formal muy poderosa para la especificación formal de consultas sobre aplicaciones hipermediales orientadas a objetos, y es absolutamente independiente del lenguaje de consulta.

Por último, la implementación del lenguaje OOHQL como extensión al OO-Navigator Framework proporciona una herramienta más, y muy poderosa, para el desarrollo de aplicaciones hipermediales orientadas a objetos. El proceso de desarrollo consiste en utilizar OOHDM como metodología de diseño, implementar dicho diseño utilizando el OO-Navigator y definir estructuras navegacionales de alto nivel, como los Contextos Navegacionales, utilizando OOHQL.

8.2 Trabajo Futuro.

En esta sección vamos a presentar algunos aspectos que fueron mencionados durante el desarrollo de esta tesis, pero que fueron dejados fuera de la implementación del sistema y que constituyen el trabajo futuro a realizar, que continúa la línea de investigación sobre este tema.

Actualmente, estamos extendiendo la implementación de OOHQL para soportar las consultas estructurales que fueron descriptas en el Capítulo 4. Creemos muy importante la posibilidad de realizar este tipo de consultas, ya que son una herramienta de mucha ayuda para el diseñador, no solo en la fase de desarrollo de la aplicación hipermedial, sino también durante la etapa de mantenimiento.

Durante este trabajo nos hemos referido, aunque no muy extensamente, al *HMQueryOptimizer*, que es el módulo encargado de realizar las optimizaciones sobre la expresión OOHA retornada por el *HMQueryParser*. Estamos estudiando la forma adecuada de realizar estas optimizaciones, determinando que reglas de optimización serán aplicadas, como difieren dichas reglas de las reglas de optimización algebraica existentes para álgebras relacionales, que otro tipo de optimización puede realizarse (por ejemplo a nivel físico), y cual es la forma correcta y más flexible de implementar dicho componente.

También hemos pensado en la posibilidad de enriquecer las herramientas visuales del QSC, proveyendo alguna forma de realizar Query-By-Example. Nuestra intención es proveerle al usuario las dos alternativas: realizar consultas mediante las herramientas descriptas en el Capítulo 7, es decir, escribiendo la consulta en OOHQL, o mediante una herramienta de composición visual de la consulta, que luego genera automáticamente el string OOHQL y ejecuta la consulta.

Por último, debemos señalar que actualmente existe un grupo de gente trabajando en la implementación de Contextos Navegacionales, concepto que fue descripto durante esta tesis. Nosotros creemos que una de las formas más sencillas y poderosas de instanciar este tipo de estructuras de navegación, es a partir de consultas. Es por eso que estamos trabajando paralelamente pero en conjunto con este otro grupo de desarrollo, en la integración del QSC con la implementación de los Contextos de Navegación. En particular, no es difícil pensar el resultado de una consulta como un Contexto de Navegación, por lo que actualmente ambos trabajos están siendo unificados.

Apéndice A

En esta sección será mostrado el ejemplo completo que fue desarrollado especialmente para esta tesis. Se han incluido todos los diagramas y especificaciones necesarias para su total comprensión, así como también los documentos desarrollados durante el diseño de la aplicación hipermedial.

1. Especificación de la aplicación

Se modelo una aplicación hipermedial destinada para un comercio de venta de libros.

El hipermedia tiene como objetivo mostrar de manera hipermedial los libros para su comercialización, la aplicación debería ser de utilidad para encontrar el libro adecuado.

Los libros tienen datos propios como, titulo, autores, precio, editorial, ISBN, y el tema principal que trata.

Cada libro tiene un conjunto de comentarios realizados por los clientes, un cliente es aquella persona que compró el libro y realiza un comentario acerca de este, el comentario tiene una fecha, el nombre y dirección de e-mail de la persona.

Los libros tienen temas cada tema tiene una lista de libros que lo tratan.

De la editorial se conoce el nombre, la dirección y la lista de libros que ella edita. Un autor también es una entidad importante, se conoce el nombre y una lista de libros que escribió.

2. Diseño OOHDM de la aplicación

Presentaremos a continuación los documentos que fueron generados durante el desarrollo de la aplicación hipermedial. Hemos desarrollado el modelo de la hipermedia utilizando OOHDM (Object Oriented Hypermedia Design Metodology), esta metodología esta conformada por cuatro etapas, en esta sección se presentaran los documentos pertenecientes a las dos primeras etapas.

Se puede apreciar un resumen de la metodología en el siguiente cuadro (cuadro 1), dicho cuadro también es incluido en el capítulo 5.

Etapas	Productos	Mecanismos	Diseños Concernientes
Diseño Conceptual	Clases, subsistemas, relaciones, perspectivas de atributos.	Clasificación, composición, generalización, y especificación.	Modelar la semántica del dominio de aplicación.
Diseño Navegacional	Nodos, Link, estructuras de acceso, contextos navegacionales	Mapeo entre el modelo conceptual y objetos navegacionales	Enfasis sobre aspectos cognitivos. Toma en cuenta perfiles de usuario y tareas.
Diseño abstracto de Interface	Objetos abstractos de interface, responsabilidades de eventos externos, transformación de interfaces	Mapeo entre objetos navegacionales y objetos perceptibles.	Modelo de objetos que se pueden percibir, elegir metáfora de implementación. Describe interface para objetos navegacionales.

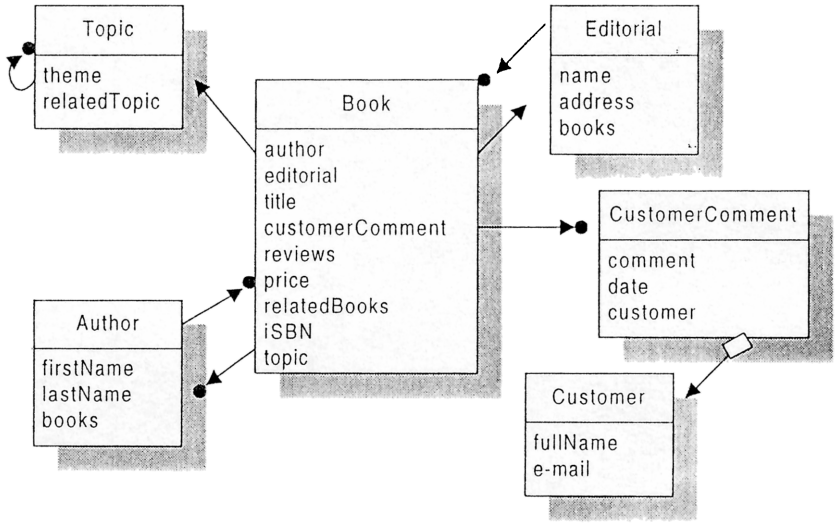
Implementación	Ejecutar la aplicación	Proveer un medioambiente	Performance, Completitud

Cuadro 1. Resumen de la metodología OOHDM

2.1. Diseño conceptual

Mostraremos el diagrama del modelo Orientado a Objetos que utilizamos para la aplicación, dicho modelo pertenece a la primer etapa del desarrollo del hipermedia.

Como se puede apreciar en la figura 1, el diagrama de clases de nuestra aplicación es sumamente sencillo. Esto es debido a que es de nuestro interés una aplicación de fácil comprensión, con el motivo de que los esfuerzos de comprensión de los lectores de la tesis se concentren en sobre el lenguaje de consultas que es lo que motivo la tesis.



+

Figura 1. Modelo Orientado a Objetos de la Aplicación

Este diagrama fue desarrollado utilizando una notación UML-Like. Los cuadros representan clases, las flechas representan una relación de conocimiento, el punto negro implica una relación uno a muchos, y el rombo de una relación indica una relación de parte_de.

2.2. Diseño Navegacional

En esta sección se presentan dos diagramas igualmente importantes.

El primero es un mapeo a nodos y links del modelo de la aplicación (Figura 2), este diagrama es esencialmente importante para esta tesis ya que el lenguaje de consultas utiliza dicho diagrama como modelo de datos. Este diagrama debe ser conocido por el usuario a la hora de consultar información, surge en el diseño OOHDM, y es el que se definió como modelo de datos en la sección cuatro de la tesis.

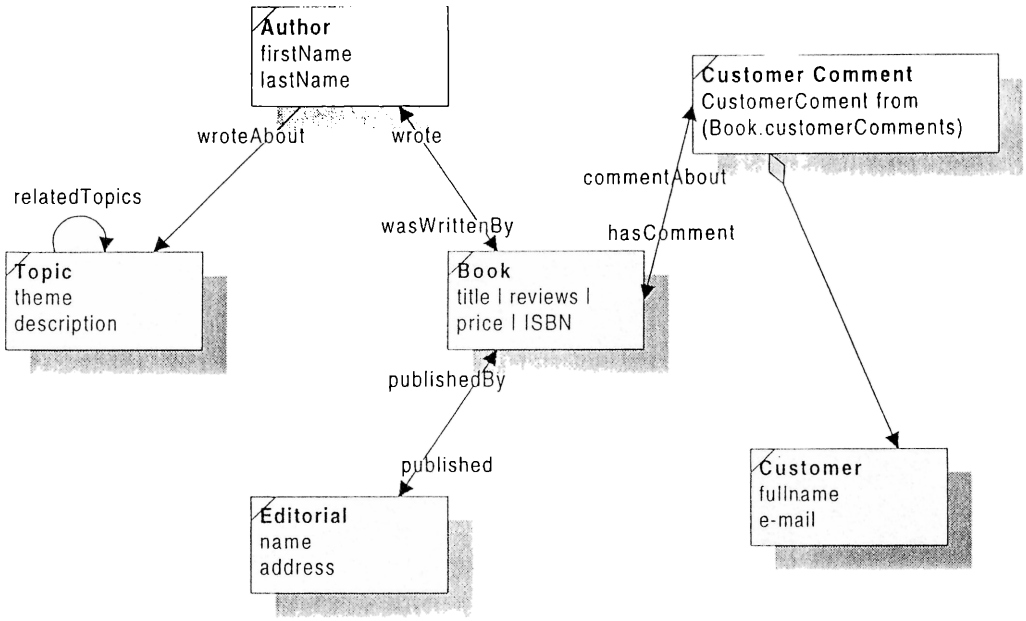


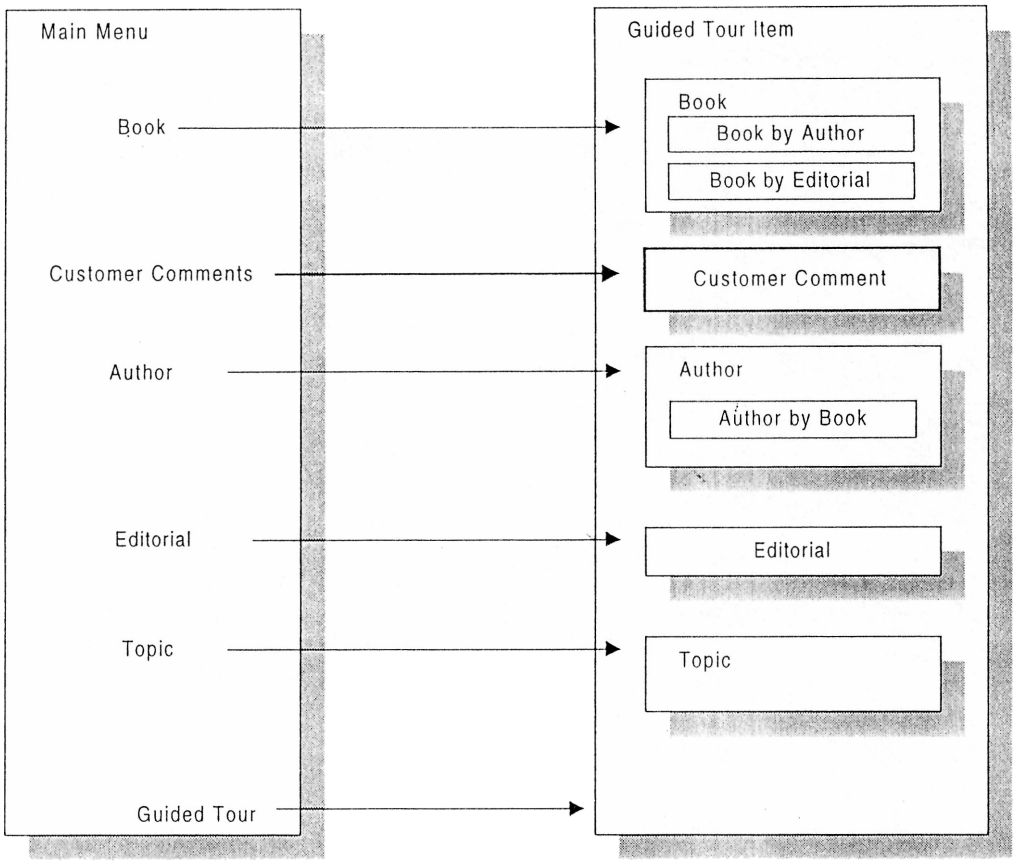
Figura 2. Nodos y Links

Este diagrama también se realiza utilizando una notación UML-Like. En este caso los cuadros indican nodos y las flechas indican Links. Las flechas con rombo describen una relación de parte_de.

El segundo diagrama que presentaremos el esquema navegacional de la aplicación (Figura 3). Dicho diagrama utiliza los contextos navegaciónales como primitivas de alto nivel para definir como se realizara la navegación de los nodos en el hipermedia.

Los conceptos sobre contextos navegacionales fueron expuestos como herramientas de navegación en el capítulo 2 de esta tesis.

Luego del diagrama en el cual se representan los contextos navegacionales, se encuentra la definición exacta de cada contexto.



Book by Editorial
∀b e Book,∃ t e Editoriall(t,b) e PublishedBy
Entry point: First Node
Path: List
Behavior: step by step

Author by Book
∀b e Author, ∃t e Book I (t,b) e wasWriting
Entry point: First Node
Path: List
Behavior: step by step

Book by Author
∀b e Book, ∃t e Author I (t,b) e wrote
Entry point: First Node
Path: List
Behavior: step by step

Editorial
∀b e Editorial
Entry point: First Node
Path: List
Behavior: step by step

Book
∀b e Book
Entry point: First Node
Path: List
Behavior: step by step

Topic
∀b e Topic
Entry point: First Node
Path: List
Behavior: step by step

Author
∀a e Author
Entry point: First Node
Path: List
Behavior: step by step

Customer Comment
∀c e CustomerComment
Entry point: First Node
Path: List
Behavior: step by step

3. Consultas

Ahora presentaremos las consultas que se encuentran desarrolladas en la tesis, el objetivo es poder contar en forma completa con los ejemplos propuestos en la tesis.

Presentaremos las consultas en su totalidad, comenzando por la especificación y concluyendo con sus representaciones algebraicas.

Ejemplo.1) Seleccionar todos los autores con nombre Orfali Robert, donde q es una variable para almacenar el resultado.

```
Q : = SELECT Author
      FROM _a: Author
      WHERE  (_a name = 'Orfali Robert')
```

$$\text{Author}[(\sigma_{[\ x\ |\ x[\text{Author}]\ \text{name} = \text{"Orfali Robert"}\ ]}\ \gamma(\text{Author}))]$$

Ejemplo.2) Seleccionar todos los Autores del libro "Java with Corba".

```
SELECT  Author
FROM  _b: Book  ,  _a : Author
WHERE  (_b title = 'Java with Corba') AND
          (Related_by(_b,wasWrittenBy,_a))
```

$$\text{Author}[(\sigma_{[\ x\ |\ x[\text{Book}]\ \text{title} = \text{"Java with Corba"}\ ]}\ \gamma(\text{Book}))\ \oplus_{x1,x2:\ (x1[\text{Book}],\ \text{wasWrittenBy},\ x2[\text{Author}])}\ (\gamma(\text{Author}))]$$

Ejemplo.3) Seleccionar todos los autores tienen libros publicados en la editorial wiley.

```
SELECT Author
FROM  _a: Author ,  _e : Editorial
WHERE  (_e name = 'Wiley') and (Path(_a,wrote,publishedBy,_e))
```

$$\text{Author}[(\gamma(\text{Author}))\ \oplus_{x1,x2:\ (x1[\text{Author}],\ \text{wrote},\ \text{publishedBy},\ x2[\text{Editorial}])}\ (\sigma_{[\ x\ |\ x[\text{Editorial}]\ \text{name} = \text{"Wiley"}\ ]}\ \gamma(\text{Editorial}))]$$

Ejemplo.4) Seleccionar todos los libros en los cuales Abiteboul sea el autor.

```
SELECT  Book
FROM  _b: Book  , _a : Author
WHERE  (_a lastName = 'Abiteboul Serge') AND
          (exist Related_by(_b,wasWrittenBy,_a))
```

$$\text{Book}[\gamma(\text{Book})\ \oplus_{x1,x2:\ (x1[\text{Book}],\ \text{wasWrittenBy},\ x2[\text{Author}])}\ (\sigma_{[\ x\ |\ x[\text{Author}]\ \text{name} = \text{"Abiteboul Serge"}]}\ (\gamma(\text{Author})))]$$

Ejemplo.5) Titulo de los libros junto al nombre de la editorial que lo edita.

```
PROJECT  Book title, Editorial name
FROM  _b: Book  , _e: Editorial
WHERE   (Related_by(_b,publishedBy,_e))
```

$\pi_{title:Book.name:Editorial}(\gamma(Book) \oplus_{x1,x2:\ (x1[Book],\ publishedBy,\ x2[Editorial])} \gamma(Editorial))$

Ejemplo.6) Todos los libros que tengan un comentario de 'Marcos Guevers'

```
SELECT Book
FROM _b: Book , _c: Customer , _cc: CustomerComment
WHERE ((_c fullName = 'Marcos Guevers') AND
          (Related_by(_b, hasComment ,_cc)))AND
          ( _c is_part_of _cc )
```

$Book[\gamma(Book) \oplus_{x1,x2:\ (x1[Book],\ hasComment,\ x2[CustomerComment])}$
$((\sigma_{[\ x\ |\ x[Customer]\ fullName\ =\ \text{"Marcos Guevers"}]}\ \gamma(Customer)$
$(\otimes_{x1,x2:\ (x1[Customer],\ x2[CustomerComment])}\ \gamma(CustomerComment)))]$

Apéndice B.
Gramática del lenguaje OOHQL.

Este apéndice tiene como objetivo presentar la definición de la gramática del lenguaje OOHQL utilizada para la implementación del analizador sintáctico durante la presente tesis. Esta gramática BNF fue implementada con un generador de intérpretes para Smalltalk denominado T-Gen [T-Gen] (Translator Generator Parser).

```
Query : 'SELECT' SelectClause 'FROM' FromClause 'WHERE'  WhereClause
|
        'SELECT' SelectClause 'FROM' FromClause |
        'PROJECT' ProjectClause 'FROM' FromClause |
        'PROJECT' ProjectClause 'FROM' FromClause 'WHERE' WhereClause
      ;

ProjectClause : ProjectClause ',' Type Message |
                ProjectClause Type Message |
                "nothing" ;

SelectClause : SelectClause Type |
               SelectClause ',' Type |
               "nothing" ;

FromClause : FromClause Variable ':' Type  |
             FromClause ',' Variable ':' Type |
             "nothing" ;

WhereClause : BoolExpr ;

BoolExpr : BoolExpr2 | '(' Term ')' ;

BoolExpr2 : '('BoolExpr2')' Boollean_Op Term2 |
            Term2 Boollean_Op '('BoolExpr2')' |
            Term2 Boollean_Op Term2 |
            '('Term2')' Boollean_Op '('Term2')' ;

Term2 : '(' Term ')' ;

Term : Field_Id Relational_Op Field_Id |
       Field_Id Relational_Op Variable |
       Field_Id Relational_Op Constant |
       ExpHyper ;

Relational_Op : '=' | '<' | '>' | '~=' ;

Boollean_Op : 'OR' | 'or' | 'AND' | 'and' ;
```

Field_Id : Field_Id Message | Variable Message ;

ExpHyper : Part_Of | Related_By | Path ;

Part_Of : Variable 'is_part_of' Variable ;

Path : 'Path('Variable','LinkTypes',' Variable')' ;

LinkTypes : LinkTypes Type {toCollection:addType:} |
LinkTypes ',' Type |
"nothing" ;

Related_By : 'Related_by(' Variable ',' TypeLink ',' Variable ')' ;

Numero : <numero> ;

String : '!' StrList '!' ;

Str : <name>;

StrList : StrList Str | StrList Numero | "nothing" ;

Constant : String | <numero> ;

Variable : <var> ;

Type : <name> ;

TypeLink : <name> {answerCollectionWith:} ;

Message : <name> ;

Bibliografía

[573991] ISO/IEC JTC1/SC21 N 5739. Database Languaje SQL, 1991.

[Abiteboul95] Abiteboul, Hull and Vianu 1995 *"Fundation of databases"* Addison-Wesley Publishing Company

[Amann92] B. Amann, M. Scholl, 1992, *"Gram: A graph data model and query language"*.Proceedings of the 4th ACM Conference on Hypertext and Hypermedia (ECHT'92), Milano, Italy, December

[Amman94] B. Amann, M. Scholl, 1994, *"Quering typed hypertexts in Multicard/O2"*. Proceedings of the 5th ACM Conference on Hypertext and Hypermedia (ECHT'94), Edinburgh, September.

[C+76] D.D. Chamberlin et al. Sequel 2: *"A unified approach to data definition, manipulation and control"*. IBM. Research and Development, 20(6): 560-575, 1976

[Campbell88] Campbell B. And Goodman J. M. 1988 HAM *"A general purpose hypertext abstract machine"*. Communiacatins of the ACM 31, 7 july, 856-861

[Campbell91] R. Campbell, N. Islam, R. Johnson, P. Kougiouris, and P. Madany. *"Choices, Frameworks and Refinement"*. In Luis-Felipe Cabrera and Vincent Russo, and Marc Shapiro, editor, Object-Orientation in Operating Systems, pages 9-15, Palo Alto, CA, October 1991. IEEE Computer Society Press.

[Codd70] E. F. Codd. *"A relational model of data for large shared data bank."* Comm. Of The ACM, 13(6): 377-387, 1970

[Concens90] M. P. Concens, A. Mendelzon, 1990, *"Gaphlog: a visual formalism for real life recursion"*. Proceedings of the ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, pp 404-416, Nashville, Tennessee.

[Diaz97] Diaz Alicia, Gordillo Silvia.1997 *"Designing Navigational Contexts using an OO Query Language."* Proceedings of DEXA'97. QPMIDS Workshop,Toulouse, France, September 1-5.

[Gamma95] Erich Gamma, Richard Helem, Ralph Johonson, John Vlissides. "Disegn Patterns. Elements of Reusable Object-Oriented Software". Addison Wesley 1995.

[Garrido96] Garrido A., Rossi G. *"A Framework for Extending Object-Oriented Applications with Hypermedia Functionality"* The New review of Hypermedia and Multimedia. Taylor Graham, vol 2, 1996

[Garrido97] Garrido Alejandra. "OO-Navigator. Un Framework para Hipermedia." Tesis de Grado. Facultad de Ciencias Exactas. UNLP

[Garzotto93]. Garzotto, P. Paolini, D. Shwabe, 1993, *"HDM - A Model Based Approach to Hypermedia Applications Design"*, ACM Trans. Info. Syst. 11, 1, January, pp 1-26.

[Goldberg83] A. Goldberg y D. Robson. Smalltalk-80 : *"The Language and its Implementation."* Addison-Wesley, Reading, Massachusetts, 1983.

[Gordillo98] Gordillo Silvia, Diaz Alicia, *"Design and Query Strategies to Hypermedia Aplication"* Multimedia Tools and Aplications 7, 213-225 (1998) Kluwer Academic Publishers. Manufactured in The Netherlands.

[Halmos60] Paul R. Halmos. *"Naive Set Theory."* D. Van Nostrand Company, INC. 1960.

[Howard95] T. Howard. *"The Smalltalk Developer's Guide to VisualWorks"*. SIGS Books, 1995.

[Isakowitz95] T. Isakowitz, E. A. Stohr, P. Balasubramaninan, 1995, *"RMM: A Methodology for Structured Hypermedia Design"*, Communications of the ACM, August.

[Johnson88] R. Johnson y B. Foote. *"Designing Reusable Classes"*. Journal of Object-Oriented Programming, 1 (2) 1988, 22-35.

[Kim90] W. Kim, 1990, *"Introduction to Object Oriented Databases"*,editado por MIT Press.

[Krasner88] G. Krasner y S. Pope. *"A cook-book for using the model-view-controller user interface paradigm in Smalltalk-80"*. Journal of Object-Oriented Programming, 1 (3), 1988, 26-49.

[kreitzberg88] Kreitzberg, C. B. And Shneiderman, B. *"Restructuring knowledge for an electronic encyclopedia."* Proc. Intl. Ergonomics Association 10 th Congress (Sydney Australia, 1-5 August 1988), 615-620

[LaLonde89] W. LaLonde, 1989, *"Designing Families of Data Types Using Exemplars"*, ACM Toplas, April

[Marmann92] M. Marmann and G.Schlargeter. *"Towards a Better Support for Hypermedia Authoring: The HYDEDIGN Model"* Proceeding of the 4th ACM Conference on Hypertext and Hypermedia (ECHT'92), Milano, Italy, December

[Mihaila96] G.A. WebSQL- *"An Sql-like query languaje for the word wide web."* Master's Thesis, University of Toronto, 1996.Canada.

[Nanard91] J. Nanard, M. Nanard, 1991, *"Using structured types to incorporate knowledge in hypertext"*. Proceedings of Hypertext'91, pages 329-343, December.

[Nilsen94] Nielsen Jakob, 1994 *"Multimedia & Hypertext: The internet and beyond"* editado por AP Professional.

[Opdyke90] W. Opdyke y R. Johnson. *"Refactoring: an aid in designing application frameworks and evolving object-oriented systems"*. Proceedings of Symposium on Object-Oriented Programming Emphasizing Practical Applicarions (SOOPPA), 1990.

[Oszu95] Oszu M. Tamer, Nlakeley Jose A, *"Query Processing in Object-Oriented Database Systems"*. Modern Database Systems. Won Kim. Addison Wesley 1995.

[Parunak91] H. Van Dyke Parunak, 1991, *"Don't link me in: Set based hypermedia for taxonomic reasoning"*. Proceedings of Hypertext'91, pages 233-242, December.

[PP94] ParcPlace Systems, VisualWorks User's Guide, 1994.

[Rational97] "UML", http://www.rational.com/uml

[Rossi97] Rossi G., Schwabe D. and Garrido A.: "Design Reuse in Hypermedia Applications Development." Proceedings of ACM International Conference on Hypertext (Hypertext'97), Southampton, April 7-11, 1997, ACM Press.

[Rumbaugh91] J.J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, 1991, *"Object Oriented Modeling and Design"*. Prentice Hall Inc.

[Schwabe95] D. Schwabe, G. Rossi, S. Barbosa, 1995, *"Systematic Hypermedia Design with OOHDM"*. Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences, Maui, Hawaii, Jannuary.

[Schwabe96] Schwabe D.. Rossi G.. Barbosa S.. *"Systematic Hypermedia Design with OOHDM."* Proceedings of the ACM International Conference on Hypertext, Hypertext'96. pp 116-128

[Schwabe98] Schwabe D. and Rossi G.: *"An object-oriented method for developing web information systems"*. Theory and Practice of object systems.October 1998

[Steenhagen96] Hennie J. Steenhagen, Rolf A. De By, Henk M. Blanken: *"Translating OSQL-Queries into Efficient Set Expressions"*. EDBT 1996: 183-197

[Shneiderman89] Shneiderman B., Brethauer D., Plaisant C.,y Potter R. 1989. *"The Hyperties electronic encyclopedia: An evaluation based on three museum installations"*. J.American Society for Information Science 40, 3 May 172-182.

[T-Gen] http://st-www.cs.uiuc.edu/users/droberts/tgen2.2.1/tgen.html.

[Woolf94] B. Woolf. "*Improving dependency notification*". The Smalltalk Report, Vol. 4 (3), SIGS Publications, Noviembre 1994.

[Yu98] Clement Yu, Weiyi Meng. "*Principle of Database Query Processing for Advanced Aplication*". Morgan Kaufmann Publishers, Inc. 1998

Artículos publicados

[Sansano99] "Query Execution on an Object Oriented Hypermedia System" Sansano Mauricio, Arambarri Federico, Diaz Alicia, Gordillo Silvia. MISRM'99 October 30 1999. Orlando, Florida. Multimedia Intellifent Storage and Retrivial Management in conjuntion with ACM Multimedia Conference 1999.

[Rossi98] "Querying Hypermedia Applications in an Object-Oriented FrameworkRossi Gustavo,Diaz Alicia, Gordillo Silvia, Sansano Mauricio, Arambarri Federico, International Workshop on Issues and Applications of Database Technology (IADT'98), pp 400-407. ISSN: 1090-9398.Ed. Society for Design and Process Science Berlin, Germany. Julio 6-9, 1998.

[Arambarri98] "Un modelo Orientado a Objetos para la implementaci¢n de OOHQL" Arambarri Federico, Sansano Mauricio ,Gordillo Silvia, Diaz Alicia, IDEAS'98 pag. 65 de los anales, congreso realizado en Torres,Brasil,abril 1-3 1998.