TRABAJO DE GRADO

UNA HERRAMIENTA CASE PARA DISEÑO DE HIPERMEDIA BASADA EN MODELOS

Directores: Lic. Gustavo Rossi
Lic. Alicia Díaz

Alumnos: Federico Damián Hermo
Andrés Leandro Blotto

Año: 1996

Tabla de contenidos

Una Herramienta CASE para Diseño de Hipermedia basada en Modelos

Federico D. Hermo y Andrés L. Blotto

RESUMEN

Nosotros presentamos una herramienta CASE para la construcción de hipermedias basada en OOHDM (Object-Oriented Hypermedia Design Model). Esta herramienta permite realizar los pasos: 2 (diseño navegacional), 3 (diseño de interfaces), y 4 (implementación) de la metodología OOHDM [Schwabe94b]; buscando facilitar el proceso de desarrollo de grandes sistemas hipermediales y aprovechando las ampliamente conocidas ventajas de usar modelos de diseño para crear aplicaciones hipermedia [Garzotto93]. Permite también transformar un sistema orientado a objetos tradicional en hipermedial.

Si bien un diseño en OOHDM puede ser implementado en los distintos sistemas de autoría (Toolbook, HyperCard, KMS, Guide, etc.), hacerlo en nuestra herramienta resulta más directo y amigable.

1. INTRODUCCIÓN

La complejidad de los dominios de las aplicaciones hipermediales (Legislación, Ingeniería, etc.), la navegación a través de un espacio de información y la dificultad de tratar con datos multimediales (en los cuales pueden ocurrir complejas transformaciones), construir la interface de la hipermedia para aplicaciones grandes; son los principales problemas con los que uno se enfrenta al construir aplicaciones hipermediales.

Estas desventajas pueden ser superadas con un buen modelo de diseño [Garzotto91]. Éste provee un lenguaje con el cual un analista puede *especificar* una aplicación determinada ayudando a la comunicación entre el analista y el usuario final; entre el analista y el diseñador y entre el diseñador y el programador. Los modelos de diseño pueden ser usados para documentar la aplicación facilitando el mantenimiento, y la comparación de diferentes aplicaciones en un lenguaje común.

Los modelos de diseño proveen un ambiente de trabajo en el cual los autores de aplicaciones hipermedias pueden desarrollar, analizar y comparar metodologías de diseño en un alto nivel de abstracción. Este análisis puede ser hecho sin la necesidad de una visualización a las pantallas, a la funcionalidad de los botones, etc.; o a los contenidos detallados de una unidad de información. Es posible analizar la organización conceptual de una aplicación en un alto nivel de abstracción.

Otra ventaja es que las aplicaciones desarrolladas de acuerdo a un modelo resultarán en una estructura de representación muy predecible. Por lo tanto, el ambiente de navegación para los lectores debería ser predecible también, disminuyendo el problema de desorientación y sobrecarga de información.

En la mayoría de los sistemas de autoría (Toolbook, HyperCard, KMS, Guide, etc.) se debe construir instancia por instancia, debiendo tener presente toda la

información concerniente a cada una de ellas; nuestra herramienta puede ser usada con un modelo conceptual previamente desarrollado en Smalltalk y luego durante la implementación de la hipermedia, debido a que hay un diseño previo, con solo identificar la instancia el resto de los datos de la misma serán "levantados" automáticamente. Así también cuando se define un nodo navegacional, se diseña cuáles serán las *perspectivas* del mismo (molde general de cómo se "ve" el nodo navegacional), luego construir la hipermedia es más fácil, basta con señalar cuál será la perspectiva para cada nodo navegacional que se vaya agregando.

En los sistemas tradicionales cada aplicación nueva deberá desarrollarse desde el principio donde la única reusabilidad posible es la técnica de *copiado y pegado.* En cambio, con nuestra herramienta, cada nodo navegacional diseñado es completamente reusable para otras aplicaciones, ya que el mismo se diseña en forma independiente de la aplicación en la que se lo define.

En nuestra herramienta los nodos navegacionales se definen jerárquicamente mediante la relación *es-un*, lo que permite el mecanismo de abstracción Generalización / Especialización y la herencia de estructura y comportamiento entre los nodos que forman esta jerarquía. Por ejemplo, si en una aplicación tenemos los nodos *Alumno* y *Profesor* podemos generalizar estos conceptos en un nodo *Persona* que reúna estructura y comportamiento común a ambos.

Otras diferencias importantes con el resto de los sistemas de autoría existentes son el uso de links tipados, links con atributos y generación automática de índices con filtros por predicado. También se proveen herramientas de navegación prácticamente automatizadas, tales como tour guiado, marcar nodos visitados, índices globales. Posibilidad de distintos lectores en una misma hiperbase. (ver Características de la Herramienta).

Nosotros presentamos una herramienta CASE basada en OOHDM, intentando facilitar el proceso de desarrollo de aplicaciones hipermedia. Esta herramienta tiene las ventajas de estar basada en modelos [Garzotto91] y de ser orientada a objetos [Schawbe94]. Proveemos un ambiente de software que permite soportar todo el ciclo de vida y a la vez permite la conexión con modelos de objetos existentes en implementaciones particulares. Esta herramienta soporta los pasos más importantes de OOHDM demostrando la viabilidad de un enfoque de este tipo para construir aplicaciones hipermedia.

La estructura de este trabajo es la siguiente: En la sección 2 describimos brevemente OOHDM y sus pasos. Luego continuamos con la descripción de la herramienta acompañada de un ejemplo, secciones 3 y 4. En la Sección 5 profundizamos en sus características principales. Más adelante, en la sección 6, presentamos la implementación; por último en las secciones 7 y 8 comentamos las futuras extensiones de la herramienta y las conclusiones. También acompañan el trabajo un manual de usuario (apéndice A) y el diseño detallado (apéndice B).

2. OOHDM. BREVE RESUMEN

2.1. Proceso de Construcción de Aplicaciones Hipermedias

OOHDM es un proceso de cuatro pasos que es una mezcla de los siguientes estilos de desarrollo: iterativo, incremental y basado en prototipos [Schwabe94b, Schwabe95, Rossi95a, Rossi95b]. En cada paso el modelo es construido o enriquecido; y después del último paso, nosotros tenemos suficiente información para implementar la aplicación hipermedia.

El siguiente cuadro resume los cuatro pasos de este proceso.

Pasos	Productos	Mecanismos	Incumbencias del Diseño
	Clases, Subsistemas, relaciones, tipos de los atributos.	Clasificación, composición, generalización y especialización.	Modelar las semánticas del dominio de la aplicación
	Nodos, Links, estructuras de acceso, contextos navegacionales, transformaciones navegacionales.	Mapeo entre objetos conceptuales y navegacionales	Se tiene en cuenta la tarea a desarrollar por el usuario final.
	Objetos de la interface, respuestas a eventos externos, transformaciones de la interface.	Mapeo entre objetos navegacionales y perceptibles.	Modelar objetos perceptibles. Describir la interface para objetos navegacionales.
	Aplicación final.	Aquellos provistos por el ambiente.	Perfomance, completitud.

2.1.1. Paso 1: Diseño Conceptual

En este paso se describe el dominio de la aplicación usando las primitivas de OOHDM para crear una hiperbase donde las contenidos básicos son organizados.

Las siguientes actividades son llevadas a cabo: definir las clases, subsistemas y relaciones de acuerdo a la semántica del dominio; construir las jerarquías de clases (usando herencia) y *parte-de*; refinar clases y asignar tipos a todos los atributos.

En OOHDM un esquema hipermedial orientado a objetos es construido en base a objetos, clases, relaciones y subsistemas.

El esquema consiste de un conjunto de objetos y clases conectadas por relaciones; los objetos son instancias de clases, y así, cuando se establece una relación entre clases, en realidad es una abstracción de una relación entre objetos. Las clases pueden estar agrupadas en subsistemas (abstracciones de un completo esquema de hipermedia).

Como es usual en los modelos orientados a objetos, las clases son descriptas mediante un conjunto de atributos tipados y métodos, y organizadas en jerarquías *es-un* y *parte-de*.

2.1.1.1. Modelando Constructores: Clases, Relaciones y Subsistemas

El producto que se obtiene en el paso de modelaje es similar a el que se obtiene usando una metodología orientado a objetos como la ORM (Object Relationship Model) de Rumbaugh [Rumbaugh91]. Sin embargo, la implementación de los objetos y las relaciones será diferente a la de las actuales metodologías orientadas a objetos. Primero, las relaciones en este modelo son abstracciones de familias de links hipermediales (entre objetos o sus partes), con la intención de ser navegadas

por el usuario final; por ello, ellas serán, en general, implementadas como links en un ambiente hipermedia.

Aunque las clases representan abstracciones de conceptos del mundo real, decidir cuáles entidades serán nodos en la hipermedia final, requiere algunas decisiones de diseño. Por ejemplo, se puede construir un nodo combinando características de diferentes clases, para representar estructuras de agregación.

Los nodos se definirán (después de decidir cuáles de los objetos del modelo se convertirán en nodos) durante el paso 2 (diseño navegacional) describiendo contextos y semántica navegacional.

Definir correctamente los atributos para las clases y las relaciones es sumamente crítico y pueden usarse heurísticas de modelos orientados a objetos ya existentes. Las distintas formas en que se pueden definir los objetos, en cuanto a la arbitrariedad de qué constituye un atributo, llevan al caso en que se debe representar relaciones entre objetos y valores de sus atributos.

Es decir, desde el punto de vista hipermedial es importante asegurarse que, cuando es posible, las relaciones no deben estar escondidas en atributos de las clases. Esto significa que cuando un atributo representa un entidad conceptual compleja con la intención de ser explorada en la hipermedia final, una relación debe ser especificada. En otros palabras, los atributos deben ser, cuando sea posible, atómicos.

En la figura 1, *Artista* podría haber sido definida como un atributo de *Monumento* (creador). Esto podría ser una correcta decisión de modelaje solamente si uno quiere desenfatizar el *Artista* como un objeto (por ejemplo, si solamente se está interesado en el *nombre* del artista). Ni tampoco debería haber atributos escondidos en relaciones. En el mismo ejemplo, el nombre del monumento es propiedad de *Monumento* y no una relación entre *Monumento* y *String*.

Figura 1. Un ejemplo de relación entre dos clases

Las relaciones son también definidas como clases que incluyen atributos y comportamiento. Los atributos describen la relación en si misma. En la figura 1, por ejemplo, la relación *creado por* puede contener un atributo fecha de creación. En aplicaciones complejas una relación puede ser aún mapeada en un objeto perceptible, si uno está interesado en visualizar sus características.

La cardinalidad de la relación es también especificada en el esquema. Una relación con cardinalidad mayor o igual que cero, indica que la relación puede no existir. Una relación n-aria es especificada indicando su clase origen y su clase destino.

Los subsistemas representan esquemas completos de una hipermedia que son usados como parte de otro esquema de hipermedia. Ellos pueden ser usados cuando otro esquema de hipermedia o aplicación ya existe. Un subsistema puede ser autocontenido (no está relacionado con clases fuera de él) o no. Usualmente los subsistemas contienen *puntos de entrada*, es decir clases del subsistema que pueden ser accedidas desde clases fuera del subsistema.

OOHDM usa tarjetas de Clase, Relación y Subsistema para documentar el modelo. Las tarjetas incluyen información acerca de lo que ellas representan. Las tarjetas son fáciles de manipular y pueden ser automatizadas.

2.1.1.2. Atributos, Tipos y Perspectivas

Los atributos de las clases son tipados y representan propiedades de los objetos (el nombre de un monumento, su descripción). El tipo (o clase) de un atributo representará una relación implícita (cuando el tipo refiere a otro objeto en el esquema y por lo tanto podría originar un link hipermedia), o la apariencia visual o retórica del atributo en la aplicación hipermedia final.

Cada posible apariencia es llamada una perspectiva del atributo. OOHDM usa la misma semántica para las perspectivas que HDM [Garzotto91]; algunos ejemplos de esta característica son los siguientes: el atributo *descripción* en un Monumento puede ser visto como un texto (una descripción arquitectónica), una fotografía, o un videoclip; la descripción de una ley puede ser vista en diversos estilos retóricos: el texto oficial, una interpretación, etc.. Nótese que como es explicado en [Garzotto93] las perspectivas originarán clases de links hipermedias no explícitamente especificadas en el esquema, o sea, aquellos que conectan diferentes perspectivas de un mismo atributo (links de perspectiva en HDM). Los atributos que serán perceptibles por el usuario final se definen en el paso 2 del proceso de diseño.

2.1.1.3. Agregación y Herencia

OOHDM provee dos constructores de abstracción: Agregación y Generalización/Especialización. La primera es útil para describir Clases complejas como agregados de otras más simples y la segunda para construir Jerarquía de Clases y usar la herencia como un mecanismo de comportamiento. Además, la noción de subsistema puede ser vista como un tercer mecanismo de abstracción de alto nivel.

Las relaciones Parte-de (capítulos de un libro, escenas de una película), son descriptas usando relaciones de agregación. Agregados en una definición de una cla-

se son similares a los componentes en HDM [Garzotto91]. Existen relaciones implícitas entre un objeto complejo y sus partes (y viceversa) y entre las partes entre sí. Ellas son llamadas links estructurales en HDM [Garzotto93]. Definir estructuras de agregación bien estructuradas es importante en hipermedia porque su especificación puede ser útil cuando se definen los contextos navegacionales y las transiciones. Entender la exacta naturaleza de una agregación, es decir, qué clase de objeto complejo ella representa es importante para construir buenas estructuras navegacionales.

Las clases heredan atributos, estructuras Parte-de, relaciones y comportamiento de su superclase. En la figura 2, la relación entre *Lugar Turístico* y *Ciudad*, refleja en realidad que las instancias de cada subclase de *Lugar Turístico* están *Localizado en* una *Ciudad*, y que *Monumentos*, *Iglesias*, etc. Comparten algunos atributos (y quizás estructuras parte-de) definidas en *Lugar Turístico*.

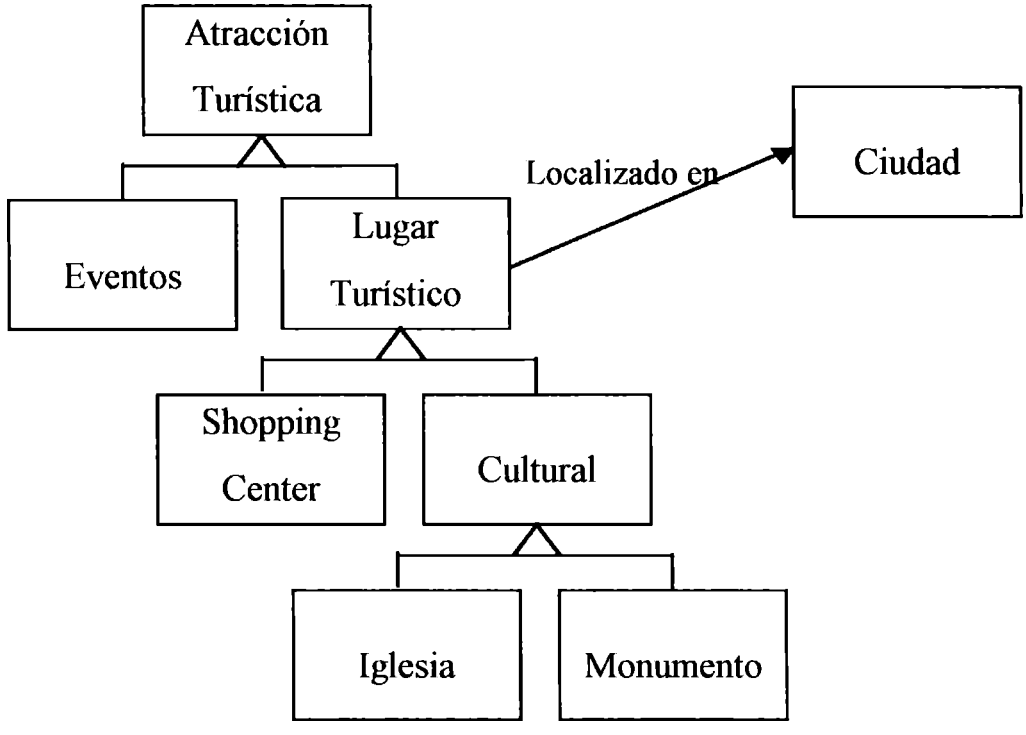


Figura 2. Un ejemplo de una jerarquía de clases mostrando la herencia de relaciones

Al menos que se indique lo contrario, Parte-de un objeto hereda los valores de los atributos del objeto padre. No está permitido la cancelación de una relación o atributo.

Consideremos el siguiente ejemplo que clarifica los conceptos explicados hasta ahora, un sistema basado en hipermedia para un departamento académico que incluye: información general del departamento, profesores, cursos, proyectos de investigación, presupuesto, información sobre fondos, publicaciones, etc.. El esquema conceptual puede verse en la figura 3.

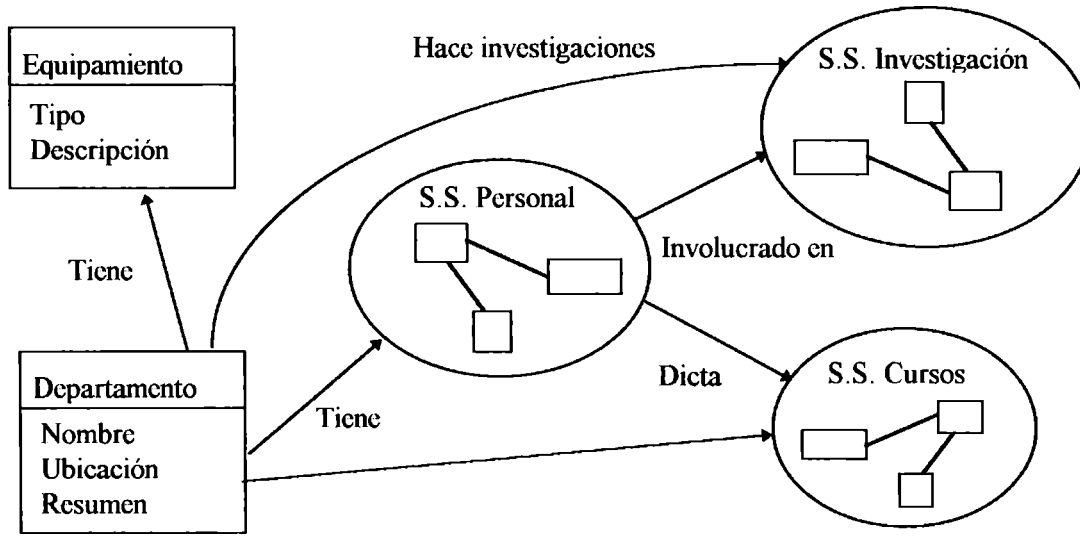


Figura 3. Esquema conceptual de un Dep. Académico

En el ejemplo se han definido dos clases principales: "Departamento" y "Equipamiento" y tres subsistemas: "Personal", "Investigación" y "Cursos". La razón para definir subsistemas no es dependiente de la aplicación sino mas bien del dominio. En la figura 4 se detalla el subsistema Personal que muestra el mecanismo de herencia, por ejemplo en la relación entre "Persona" e "Historia Personal", refleja el hecho de que las instancias de cada subclase de "Persona" regis-

tra una historia personal y que profesores, administrativos, etc. comparten atributos definidos en "Persona".

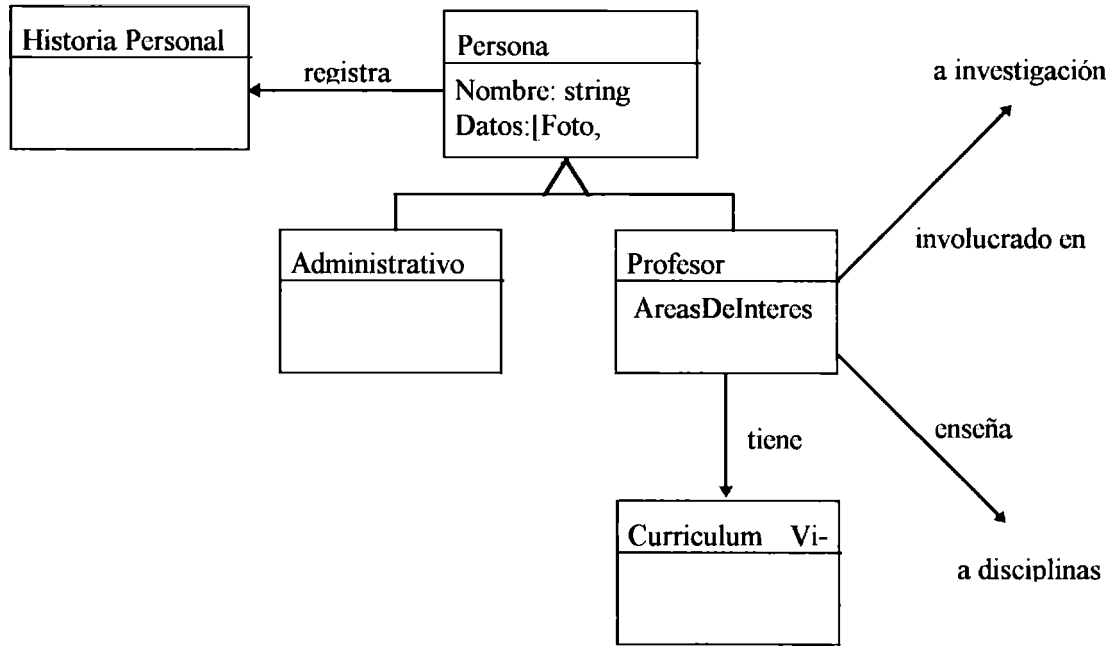


Figura 4. Esquema conceptual del subsistema Personal

2.1.2. Paso 2: Diseño Navegacional

Una de las características principales de las aplicaciones hipermedia es la noción de navegación a través de un espacio de información, por lo tanto, diseñar la estructura navegacional de estas aplicaciones es "el" paso crítico en el desarrollo. Sorpresivamente, aunque muchos autores han destacado los problemas que se originan en las estructuras navegacionales mal diseñadas [Garzotto91], los métodos de diseño de hipermedias existentes no proveen formalismos adecuados para definir el aspecto navegacional de una aplicación hipermedia.

Volviendo al ejemplo del Departamento Académico descripto en la sección anterior, éste está compuesto por varias aplicaciones que soportan diferentes tipos de usuarios, cada uno con diferentes necesidades de información. Posibles tipos de

usuarios son: visitantes (al departamento), estudiantes y agencias interesadas en subsidiar un proyecto de investigación. Debería estar claro que la información accedida por los visitantes acerca de proyectos y cursos tendrá un diferente interés del dominio que un gerente de una agencia inversionista que quiere explorar los proyectos de investigación para decidir si financia o no al departamento.

Un desarrollo tradicional trataría cada una de estas aplicaciones separadamente, aún si ellas comparten información común. Por ejemplo, un gerente y un visitante tendrán diferentes patrones navegacionales y posiblemente usen diferentes tipos de medios para ver la información (un gerente prefiere leer un breve resumen en vez de escuchar una narración de las actividades del proyecto). Claramente, se necesitaría el doble de esfuerzo en el mantenimiento, no solamente si existe la necesidad de actualizar la información, sino principalmente, si existe la necesidad de modificar cada modelo a causa de cambios en el dominio.

Otra posibilidad es construir un modelo conceptual más general que soporte ambos contextos (y quizás otros). Sin embargo, este modelo será muy complejo; primero porque debe integrar diferentes puntos de vista, luego porque incluirá abstracciones de alto nivel para proveer generalidad y flexibilidad. Para vencer la complejidad agregada, es necesario un mecanismo conceptual, que nos permita especificar qué aspectos del modelo serán usados en cada contexto.

En OOHDM, un aplicación es tratada como una vista navegacional sobre el dominio conceptual. Esta vista es construida desde la perspectiva del tipo de usuario final, y el conjunto de tareas que él podrá ejecutar. Diferentes modelos navegacionales pueden ser construidos para el mismo esquema conceptual que expresan diferentes vistas (aplicaciones) del mismo dominio.

Mientras diseñamos la estructura navegacional de una aplicación hipermedia debemos tener en cuenta los siguientes aspectos:

- Cuál es la estructura de navegación subyacente. ¿En qué contextos el usuario navegará?.

- Cuáles objetos serán navegados y cuál es su estructura, cuáles son las relaciones entre estos objetos y aquellos definidos en el esquema conceptual.

- Qué estructuras de conexión existen entre los objetos que serán navegados (links, índices, tours, etc.).

- Nosotros necesitamos decidir si los objetos navegables pueden lucir diferente de acuerdo al contexto en el cual ellos son visitados y especificar claramente estas diferencias.

Como explicamos anteriormente, en OOHDM las aplicaciones son definidas como vistas navegacionales del esquema conceptual. Se describe la estructura navegacional de una aplicación hipermedia definiendo clases navegacionales que reflejan la vista elegida sobre el dominio de la aplicación. La estructura navegacional es definida en términos de contextos navegacionales[Schwabe95], los cuales son derivados desde clases navegacionales tales como nodos, links y estructuras de acceso.

Las clases navegacionales pueden o no estar relacionadas de una forma 1 a 1 con las clases definidas en el paso 1. Para la mayoría de los casos, habrá un mapeo 1 a 1 de las clases conceptuales y relaciones con Nodos y Links en una vista relacional. En algunos casos habrá Nodos conteniendo características de más de una clase conceptual, y en otros un Nodo filtrará información (atributos y relaciones) de la correspondiente clase conceptual.

Cuando una hipermedia es muy grande o cuando ésta involucra muchas instancias de una misma clase es necesario definir estructuras de acceso[Garzotto93]. Una estructura de acceso actúa como un índice o diccionario y es útil para ayudar

al usuario final a encontrar la información deseada (una lista de teatros, de hoteles en una ciudad, un catálogo de CD, etc.). Las estructuras de acceso son caracterizadas por un conjunto de selectores, un conjunto de objetos destino (usualmente objetos en el esquema) y un predicado sobre los objetos destino. El predicado expresa qué objetos serán accesibles en términos de sus propiedades. Los selectores usualmente representan algún atributo de los objetos destino.
No toda clase del esquema conceptual del paso 1 será un nodo de una hipermedia, ni toda relación será un link; mas aún, los nodos de una aplicación hipermedia pueden contener información de más de una clase conceptual.

Las clases navegacionales representan nodos y links. Los nodos tienen atributos, anchors para links y estructuras de acceso. Los links contienen información sobre los nodos origen y destino y sus propios atributos; la cardinalidad de éstos puede ser 1 a 1 o 1 a N. Las clases navegacionales son organizadas en una jerarquía *es-un* y definen contextos para la navegación.

Es importante notar que las estructuras definidas en este paso son aún conceptuales, al mismo nivel que el domino de aplicación (y como las estructuras definidas en el paso 1).

2.1.3. Paso 3: Diseño Abstracto de Interfaces

Como previamente dijimos, todas las estructuras definidas en los pasos 1 y 2 son de alto nivel, en el sentido de estar al nivel de abstracción del dominio de la aplicación. En este paso, el autor especifica el comportamiento dinámico de la aplicación.

Las aplicaciones hipermedia son esencialmente interactivas, requieren que el autor deba especificar cuáles son los objetos perceptibles que el autor hará disponibles al usuario, y cómo ellos se comportan en términos de las acciones originadas por el usuario. Algunos de los objetos perceptibles pueden ser activados por

el usuario, y tal activación causará transformaciones en el contexto de percepción (el conjunto de objetos perceptibles). El comportamiento dinámico de la aplicación puede ser especificado como el conjunto de posibles transformaciones dado cualquier contexto de percepción.

Durante el diseño abstracto de interfaces, el autor tiene que mapear los objetos definidos en los pasos 1 y 2 en objetos perceptibles. En particular, mapeando los anchors (abstractos) en objetos perceptibles activos, esto es posible definiendo cómo la aplicación procederá cuando el usuario navega la hiperbase.

Después que este paso haya sido desarrollado nosotros tenemos suficiente información para implementar la aplicación usando un ambiente de desarrollo de hipermedias. En otras palabras, el diseño abstracto de interfaces provee el mapeo entre el diseño abstracto de alto nivel de la aplicación y la implementación concreta en un ambiente de software.

2.1.4. Paso 4: Implementación

Para mapear el diseño abstracto de interfaces (objetos perceptibles y sus transformaciones) en objetos concretos de interface (aquellos disponibles en el ambiente de implementación elegido) el autor produce el actual sistema de hipermedia para ser "corrido".

Al estar basado en un conjunto uniforme de constructores de modelo (objetos y clases), esta aproximación permite una suave transición desde el modelo conceptual al diseño navegacional y de interfaces y a la implementación. Siendo orientado a objetos, éste provee un modelo natural para seguir los pasos, lo que permitiría mejor entendimiento y mantenimiento de aplicaciones hipermedia. Esto puede también ser la base para une teoría de reuso de componentes hipermediales.

3. UNA PROPUESTA DE HERRAMIENTA

3.1. Objetivo

Lo que nosotros proponemos es una herramienta que facilite el desarrollo de aplicaciones hipermedia. Nuestra herramienta está basada en modelos, lo que "obliga" al autor a hacer un diseño antes de implementar la hipermedia, con esto, se obtiene una hipermedia más coherente, predecible y mantenible, al tomar las decisiones importantes (como la estructura de navegación) en la etapa de diseño y no mientras se está construyendo la misma. Además el modelo es orientado a objetos, aportándole a la herramienta las características básicas de este paradigma como el mecanismo de generalización/especialización, reusabilidad, facilidad de mantenimiento, etc..

3.2. Modos de Uso

Nuestra herramienta tiene dos formas principales de uso: 1) Arrancando con un modelo conceptual previamente implementado en Smalltalk; 2) Arrancando desde cero sin basarse en un modelo conceptual explícitamente desarrollado (solo hecho en papel).

En el primer caso el autor o bien desea convertir una aplicación ya existente en Smalltalk a hipermedial, o bien crea el modelo conceptual para la aplicación hipermedia antes de desarrollar la misma, para lograr varias vistas navegacionales [Schwabe95] del mismo modelo conceptual. Una vista navegacional es construida teniendo en cuenta el tipo de usuario final y el conjunto de tareas que él podrá ejecutar usando la aplicación.

A partir de las clases conceptuales se obtienen las clases navegacionales. Estas últimas pueden definirse en base a una o más de las clases conceptuales, esto se

explica con más profundidad en la sección 5. Cuando se está desarrollando el paso 4 (Implementación) del proceso de construcción de aplicaciones hipermedia de OOHDM, los datos de cada nodo de la hiperbase son tomados desde el modelo conceptual y no ingresados por el autor.

Para el otro caso, el autor puede desarrollar la aplicación hipermedia sin un modelo conceptual debajo, aunque sí es conveniente haber hecho al menos un bosquejo del paso 1 (Diseño Conceptual) del proceso, para un mejor aprovechamiento de las características de la herramienta. Durante el paso 4 (Implementación) el autor deberá completar los valores para los atributos de cada nodo que agrega a la hiperbase.

3.3. Descripción General

La herramienta está dividida en siete funciones principales: un editor de esquemas, un editor de nodos, un editor de links, un editor de subsistemas, un editor de estructuras de acceso, un editor de hiperbase y una utilidad para navegar hiperbases existentes.

El editor de esquemas sirve para tener un visión global del esquema de navegación, permitiendo un acceso rápido a los demás editores (de nodos, de links, de subsistemas y de estructuras de acceso).

El editor de nodos permite definir los nodos navegacionales, pudiendo indicar su nombre, su superclase, sus atributos y perspectivas, además de indicar con qué clase/s del modelo conceptual está relacionada, en particular, cómo se relaciona cada uno de sus atributos con los de la/s clase/s conceptual/es. Las perspectivas se definen a través de un editor de perspectivas, en este editor se diseña cuáles atributos van, dónde van, qué otros elementos se incluyen (cuadrados, líneas, texto, bitmaps, etc.), tamaño, colores, etc..

Con el editor de links se definen las relaciones entre los nodos navegacionales, indicando: nombre, superclase, atributos (si los tiene), tipo de los nodos origen y destino, cardinalidad, y también se tiene acceso al editor de perspectivas para el caso en que el link contenga atributos, donde se edita cómo se verá el nodo intermedio entre el origen y el destino.

El editor de subsistemas permite agrupar los nodos navegacionales ya definidos en subsistemas, además de indicar cuáles serán los puntos de entrada del mismo.

En el editor de estructuras de accesos se definen índices globales, indicando cuál es el nombre, el tipo de nodo destino, el selector y el predicado que filtra los destinos de acuerdo a una condición; para definirlo se incluye un editor de predicado.

El editor de hiperbase es el que permite definir la aplicación en sí, instanciando todos los elementos previamente definidos. En este paso se decide cuáles herramientas de navegación estarán disponibles al lector: botón back, botón inicio, marcas de nodos visitados, tour (sí es que se define un tour guiado para la aplicación) y estructuras de acceso globales (instancias de las definidas en el editor de estructuras de acceso). Para construir una aplicación se irán agregando los nodos y cada vez que la hiperbase es salvada se establecerán automáticamente los links para los nodos navegacionales que estén relacionados con instancias de las clases conceptuales, y sus destinos ya formen parte de la hiperbase, es decir, para el caso en que una aplicación esté basada en un modelo conceptual previamente implementado, con sólo agregar los nodos y relacionarlos con la instancia correspondiente al mismo es suficiente para crear una hiperbase. Cada nodo puede ser totalmente modificado durante esta etapa, lo único no permitido es el borrado de los objetos que corresponden a atributos para no perder el concepto de lo que el nodo representa.

La utilidad de navegación es lo que los lectores utilizarán para recorrer las hiperbases definidas.

4. EJEMPLO: Galería de Arte

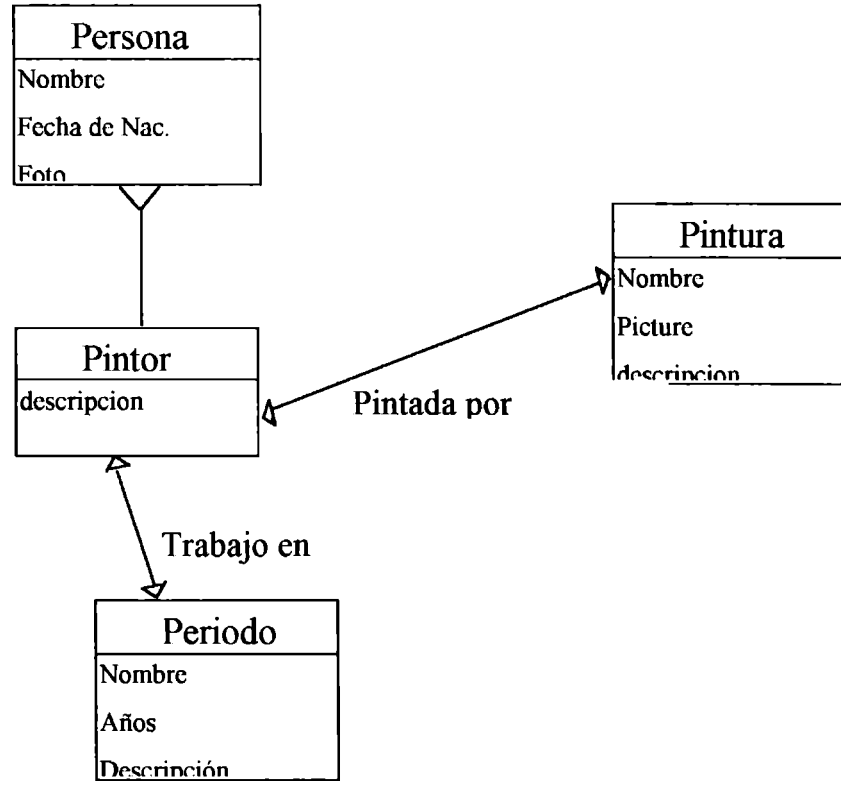
Este ejemplo que desarrollamos consiste en un pequeña galería de arte con algunos pintores y sus obras. El ejemplo que detallaremos a continuación se encuentra junto con la herramienta en el proyecto llamado *ejemplo*.

Es obvio que al público general le interesarán distintas cosas que, por ejemplo, a un estudiante de arte, el cual querrá obtener información más profunda sobre los pintores y sus obras que el usuario común que solo le interesará ver las obras y una vista por arriba de sus autores. Para esto hacemos un solo modelo conceptual que abarca los dos contextos. Luego de éste derivaremos una vista navegacional para cada contexto.

Comenzando con el paso 1 del proceso de construcción de aplicaciones de OOHDM, definimos la clases como: "Persona", "Pintor", "Pintura", "Periodo", las relaciones entre clases como "Pintó", "Trabajo en", etc. Para llevar el modelo conceptual a Smalltalk definimos las siguientes clases (las relaciones pasan a ser variables de instancia de las clase, por ejemplo, "Pintó" se refleja en la variable de instancia 'obras' de la clase Pintor) que se encuentran agrupadas en la categoría *Ejemplo*:

- **Persona**: nombre, fecha de nacimiento, foto.
- **Pintor (Persona)**: período, obras, descripción.
- **Pintura**: nombre, pintor, picture, descripción.
- **Período**: nombre, años, pintores principales, descripción.

El esquema conceptual que nos deja este primer paso es el siguiente:



Esto completaría el Paso 1. A partir de este modelo derivamos la dos vistas navegacionales nombradas anteriormente, definiendo los nodos navegacionales, links, estructuras de acceso, etc. para cada una de ellas; esta tarea se encuadra dentro del paso 2 de OOHDM. Para facilitar esta tarea, además de poder visualizar globalmente la vista navegacional, definimos dos esquemas de clases, uno para cada vista (figuras 5 y 6).

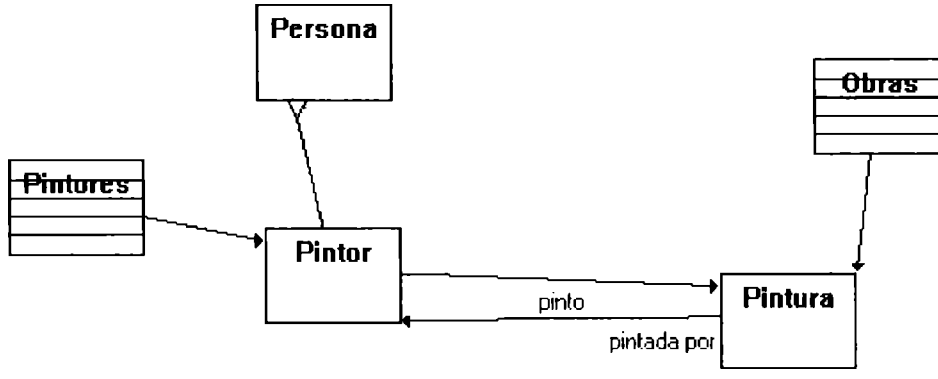


Figura 5. Esquema de clases de la vista navegacional general

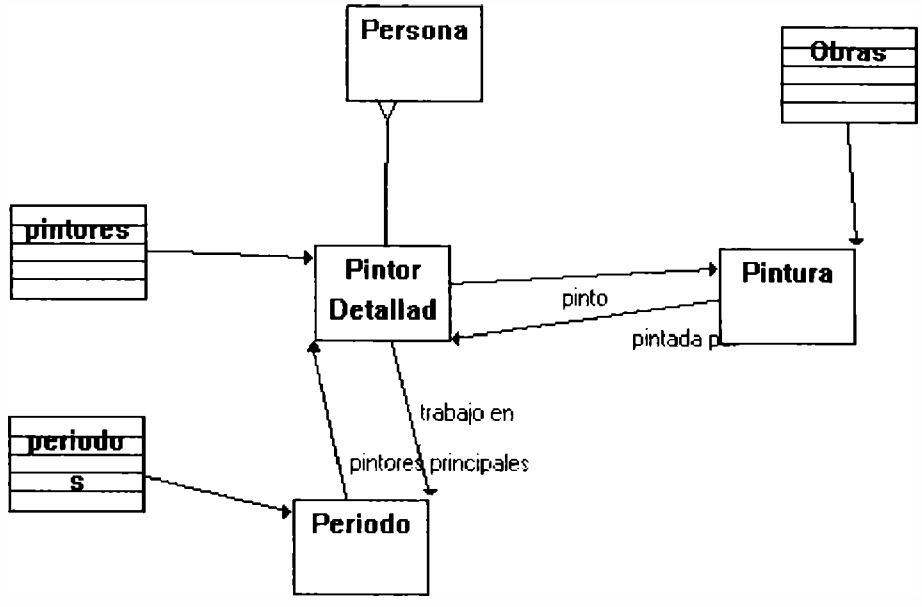
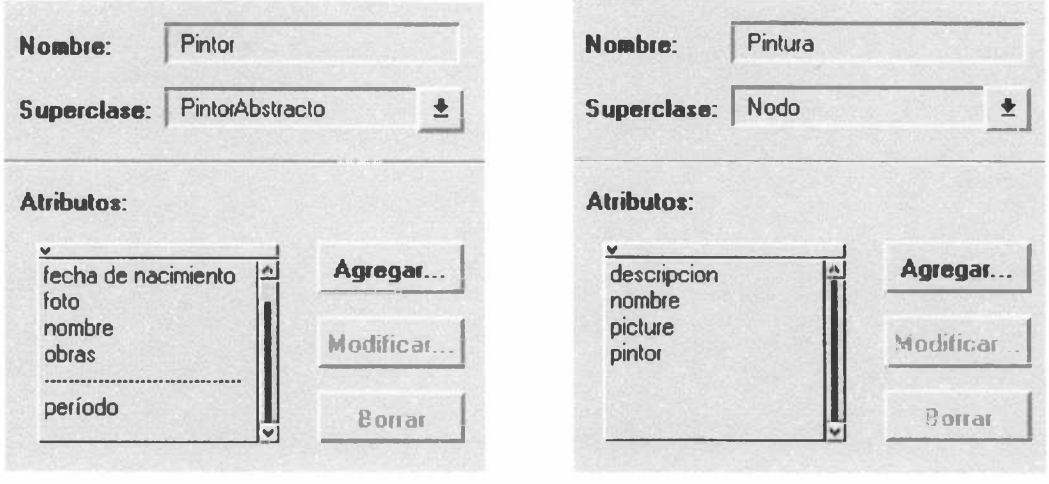


Figura 6. Esquema de clases de la vista navegacional detallada

El nodo navegacional Persona tiene los atributos nombre (String), fecha de nacimiento (Fecha) y foto (Gráfico); y es subclase de Nodo. En ambas vistas navegacionales la definición del nodo es la misma. Este nodo no tendrá instancias en la hiperbase final pero lo definiremos para que se vea como funciona el mecanismo de herencia.



(a) (b)

Figura 7. Visiones parciales del Editor de Nodos
(a) Nodo Navegacional Pintor
(b) Nodo Navegacional Pintura

Necesitaremos definir dos nodos navegacionales de la clase Pintor para las dos vistas navegacionales antes nombradas. Esto se debe a que el atributo periodo en un caso es de tipo string (solo el nombre del periodo) y en el otro es de tipo anchor (un link a un nodo Periodo). Ya que el resto de la información es igual definimos un nodo navegacional PintorAbstracto que reúne la estructura en común. Este es subclase del nodo navegacional Persona (hereda los atributos: fecha de nacimiento, foto y nombre) los atributos propios son: obras y descripción. El atributo obras es un anchor de tipo *pinto* (Ver definición mas adelante) y descripción es de tipo string. Subclase de este último son: el nodo navegacional Pintor (figura 7a) con el atributo período que es un string tomado de la clase conceptual Período a través de la relación "trabajo en"; y el nodo navegacional PintorDetallado con el atributo periodo como un anchor del tipo *trabajoEn*.

El nodo navegacional Pintura (figura 7b) es subclase del nodo navegacional abstracto Nodo. Sus atributos son: nombre (String), picture (Gráfico), descripción (String) y pintor (Anchor a Pintor); el tipo de link de este último es *pintada por* descripto más adelante. Para diferenciar las dos vistas navegacionales lo único que debemos hacer son dos perspectivas distintas del nodo Pintura. A una la llamamos "clásico" que no incluye al atributo descripción y la otra la llamamos "detallada" que sí incluye a este atributo.

En la definición de los dos últimos nodos navegacionales puede verse la dos formas de construir distintas vistas navegacionales de un mismo modelo conceptual. En el primer caso definimos dos nodos navegacionales distintos (Pintor y PintorDetallado) derivados de la misma clase conceptual Pintor. Y en el segundo caso lo único que tuvimos que hacer fueron dos perspectivas distintas de un único nodo navegacional Pintura.

También definiremos un nodo navegacional Periodo, subclase de Nodo, con los atributos nombre (String), descripción (String), años (String) y pintores (Anchor)

este anchor es del tipo *pintoresPrincipales*. Este nodo navegacional no es usado en la vista navegacional pensada para el público general, sólo es usada en la vista navegacional más detallada.

Nuestra próxima tarea es relacionar los nodos navegacionales con las clases del modelo conceptual. Las relaciones se hacen a nivel de atributo. Por cada uno que queramos relacionar debemos indicar la clase conceptual y la variable de instancia de dicha clase con la cual estamos estableciendo la relación. También debemos señalar cuál es el método (selector) que devuelve el valor de dicha variable de instancia. En caso de que estemos relacionando un atributo de tipo anchor, también deberemos indicar cuál es el método identificador del objeto destino (método que devuelva un string que permita más tarde, durante la implementación, identificar el destino para relacionar el atributo con la instancia correspondiente, y durante la navegación, individualizar a cada destino en un link de cardinalidad 1 a N).

Las relaciones con el modelo conceptual para el nodo navegacional Pintor se pueden ver en el siguiente cuadro:

Atributo	Clase	Vble. de Instancia	Selector
nombre	Pintor	nombre	nombre
fecha de nacimiento	Pintor	fechaNac	fechaNac
foto	Pintor	foto	foto
obras	Pintor	obras	obras (nombre)
período	Pintor	período	período-nombre

Todas estas relaciones las agruparemos en un único grupo, ya que todos los atributos se relacionan con la misma instancia de la clase Pintor, y con esto basta

con relacionar un atributo con una instancia en particular, para que el resto se relacione automáticamente. Para el atributo período existe otra posibilidad que es elegir la clase Período, la variable de instancia "nombre" y el selector "nombre". La opción elegida por nosotros tiene la ventaja que al poder agruparse con los demás atributos, en la etapa de implementación no tendré que recordar cuál es el período de cada pintor debido a que se deriva automáticamente del modelo conceptual.

Las relaciones para el nodo navegacional PintorDetallado son las siguientes:

Atributo	Clase	Vble. de Instancia	Selector
nombre	Pintor	nombre	nombre
fecha de nacimiento	Pintor	fechaNac	fechaNac
foto	Pintor	foto	foto
descripción	Pintor	descripción	descripción
obras	Pintor	obras	obras (nombre)
período	Pintor	período	período (nombre)

Igual que para el nodo Pintor todas las relaciones las agrupamos juntas. Si observamos la relación para el atributo periodo, en el caso del nodo navegacional Pintor el selector es período-nombre, esto significa que para acceder al valor correspondiente al atributo, a la instancia con la que está relacionado le enviamos el método periodo y al resultado el método nombre, obteniendo así un string que contiene el nombre del período al que pertenece el pintor que estamos relacionando. En cambio, para el nodo navegacional PintorDetallado, el selector es periodo (nombre), esto indica que a la instancia sólo le enviamos el método periodo para obtener el destino del anchor, y nombre es el método identificador para cuando necesitemos mostrar los posibles destinos para que el usuario seleccione con cuál de ellos relaciona el atributo periodo en el momento de instanciar un

nodo navegacional PintorDetallado en el editor de hiperbases. Además, agregamos en el nodo navegacional PintorDetallado la relación para el atributo descripción, la cual no estaba presente en el nodo navegacional Pintor porque no hacía uso de ella.

A continuación se detallan las relaciones correspondientes al nodo navegacional Pintura:

Atributo	Clase Conceptual	Variable de Instancia	Selector
nombre	Pintura	nombre	nombre
picture	Pintura	picture	picture
descripción	Pintura	descripción	descripción
pintor	Pintura	pintor	pintor (nombre)

En el siguiente cuadro podemos visualizar las relaciones para el nodo navegacional Periodo:

Atributo	Clase Conceptual	Vble. de Instancia	Selector
nombre	Periodo	nombre	nombre
años	Periodo	años	años
descripción	Periodo	descripción	descripción
pintores	Periodo	pintores	pintores (nombre)

Los links que debemos definir son: *Pintada por* con origen Pintura y destino PintorAbstracto, la cardinalidad es 1 a 1; *Pinto* con origen PintorAbstracto y destino Pintura y la cardinalidad es 1 a N; *TrabajoEn* cuyo origen es PintorDetallado, su destino Periodo y su cardinalidad 1 a 1; y *PintoresPrincipales* con origen Periodo, destino PintorDetallado y cardinalidad 1 a N. Estos dos últimos tipos de

links sólo son usados en la vista navegacional para usuarios especializados. Los links *Pintada por* y *Pinto* nombran a PintorAbstracto, ya que ambos links pueden ser usados tanto por instancias del nodo Pintor como por instancias del nodo PintorDetallado (ambos son subclase de PintorAbstracto y por el mecanismo de herencia podrán hacer uso de los links).

También definiremos cuatro estructuras de acceso: Por un lado *Pinturas*, con nodo destino Pintura y selector *nombre*; *Periodos* con nodo destino Periodo y selector *nombre*; y por otro lado, *Índice Pintores* y *Renacentistas* (figura 8). Estas dos últimas tienen como nodo destino a PintorAbstracto (pueden ser usadas tanto para instancias de Pintor como para instancias de PintorDetallado) y como selector a *nombre*. Además *Renacentistas* tiene un predicado, este es "período = Renacimiento", que filtra los pintores que no pertenecen al período Renacimiento.

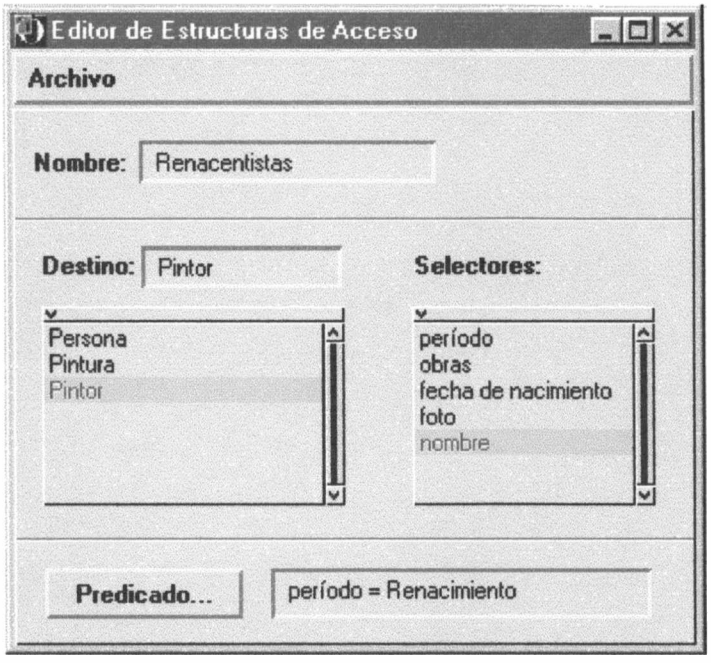


Figura 8. Estructura de Acceso: Renacentistas

El siguiente paso en el proceso de construcción de aplicaciones de OOHDM es el 3, esto es diseñar la interface de los nodos, en nuestro caso por medio de las perspectivas. Para nuestro primer ejemplo (orientado al público en general), noso-

tros diseñamos dos perspectivas para el nodo navegacional Pintor, una llamada *clásico* (figura 9) y otra llamada *sinfoto* que es igual pero sin la imagen con atributo *foto*; y una para el nodo navegacional Pintura también llamada *clásico* (figura 10).

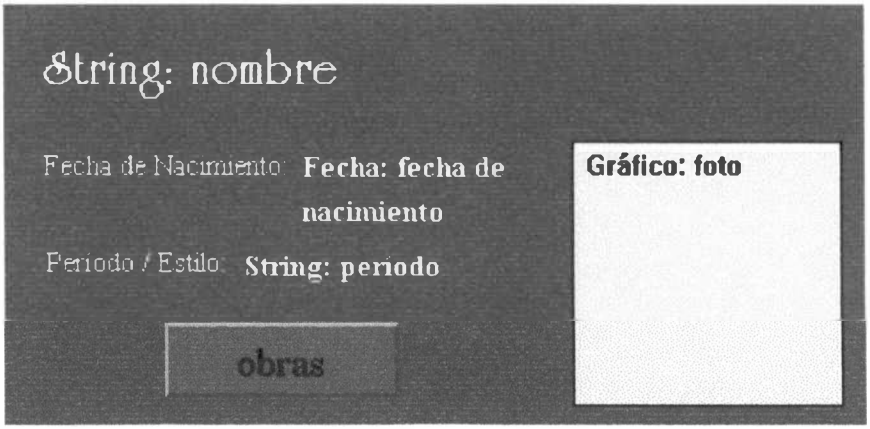


Figura 9. Interface del nodo navegacional Pintor

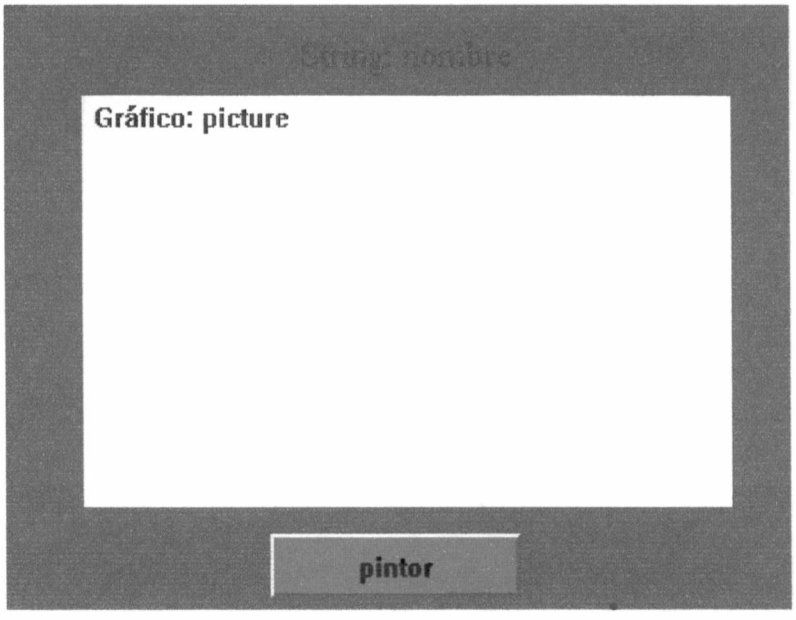


Figura 10. Interface del nodo navegacional Pintura

Para el ejemplo más específico definimos la perspectiva del nodo navegacional PintorDetallado, llamada *clásico* (figura 11). Para el nodo navegacional Pintura

definimos otra perspectiva llamada *detallada*. Y para el nodo navegacional Periodo la perspectiva llamada *detallado* (figura 12).

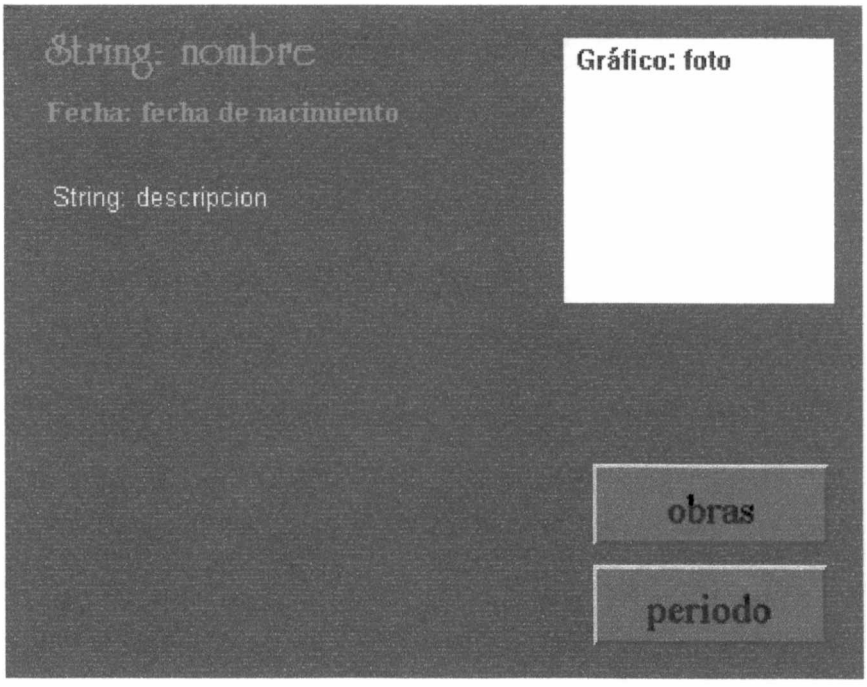


Figura 11. Perspectiva *clasico* del nodo PintorDetallado

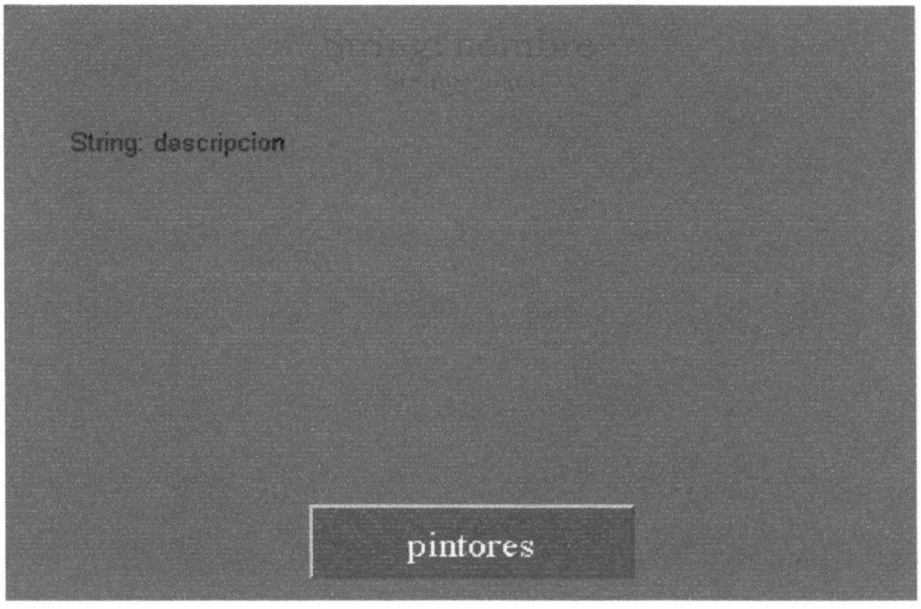


Figura 12. Perspectiva *detallado* del nodo navegacional Periodo

Por último nos queda construir la hiperbase, paso 4 (implementación) del proceso de OOHDM. Describimos a continuación como crear la hiperbase para el ejem-

plo pensado para el público no experto, el proceso es muy similar para el otro caso y no lo detallaremos.

Primero, instanciamos todos los nodos a través de la función *nuevo nodo*. Para los pintores, en el cuadro de diálogo elegimos el nodo Pintor, la perspectiva *clásico* (para instanciar un Pintor con foto) o la perspectiva *sinfoto* (para instanciar un Pintor sin foto), de la lista de relaciones elegimos *nombre* (es la que mejor identifica al objeto) y por último la instancia deseada del modelo conceptual (por ejemplo: Salvador Dalí), con este simpleza el nodo quedará instanciado y todos sus atributos contendrán los valores de la instancia elegida. En la figura 13 se ve este proceso.

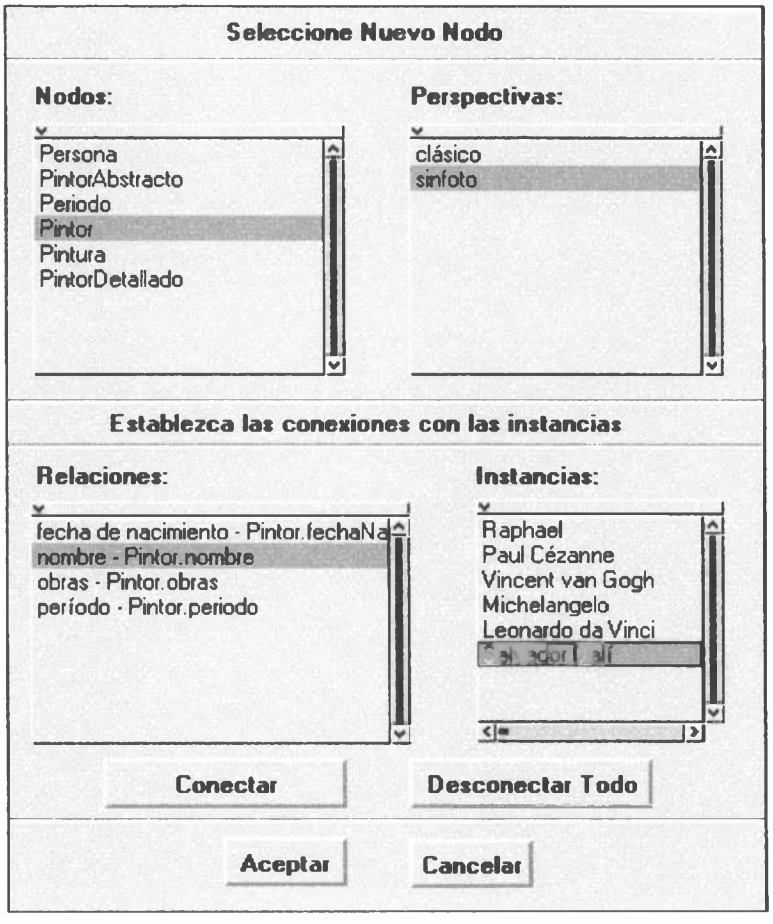


Figura 13. Interface de la función nuevo nodo.

Para las pinturas, elegimos el nodo Pintura y la perspectiva *clasico*, también elegimos *nombre* de la lista de relaciones y finalmente una instancia (por ejemplo: "La Metamorfosis de Narciso"). Una vez instanciado un nodo se pueden hacer las modificaciones que se quieran a la perspectiva original (salvo borrar atributos).

A continuación instanciamos las estructuras de acceso definidas anteriormente, por medio de la función *estructura de acceso* → *nueva*, le ponemos un nombre y solo nos queda elegir los colores y tipo de letra de la misma, el sistema automáticamente generará el índice durante la etapa de navegación con todos los nodos que cumplan con la definición de la estructura de acceso.

Para definir un tour elegimos la opción *definir tour* y uno a uno iremos agregando los nodos del mismo, por ejemplo, un tour posible es recorrer la secuencia pintor - obras - pintor - obras - etc. (Salvador Dalí - La Metamorfosis de Narciso - Sacramento de la Última Cena - Paul Cézanne - etc.).

La opción *barra de herramientas* del menú *nodo* nos permitirá marcar cuáles herramientas estarán disponibles, en nuestro caso marcamos todas ya que tenemos estructuras de acceso y un tour definido.

Luego, grabamos la hiperbase y ya estará lista para ser navegada. Los links entre los pintores y sus obras y viceversa, han quedado automáticamente establecidos; como así también, las estructuras de acceso instanciadas referencian a todos los nodos que cumplen con el tipo y el predicado. Algunos de los nodos de la hiperbase pueden verse en la figuras 14 y 15.

Figura 14. Paul Cézanne

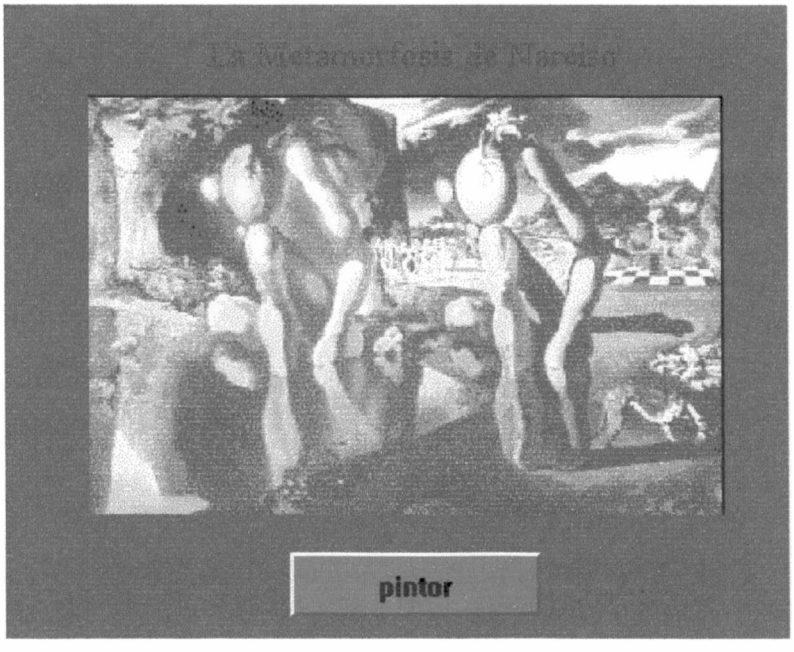


Figura 15. La Metamorfosis de Narciso (Salvador Dalí)

La navegación es muy sencilla, simples clicks del mouse bastan para navegar por la hiperbase, ya sea en los botones definidos en el nodo o usando las funciones de la barra de herramientas: inicio, back, next, previous, etc.(figura 16). Vale recordar que para recorrer el tour definido se debe comenzar desde el principio del mismo y si a mitad del tour nos apartamos del mismo usando otra herramienta de navegación, no podremos retomarlo; estas restricciones fueron tomadas para evitar el "getting lost".

Figura 16. Dos barras de herramientas

En la figura 16 se ven dos barras de herramientas de la utilidad de navegación, en la superior están presentes todas las funciones mientras que en la inferior no se encuentran las funciones next-previous y marcar nodos visitados. Estas selecciones se pueden efectuar en el editor de hiperbase por medio de la función *Nodo* ➔ *barra de herramientas*.... Nótese que en la figura superior están deshabilitados los botones next-previous ya que no se está recorriendo el tour.

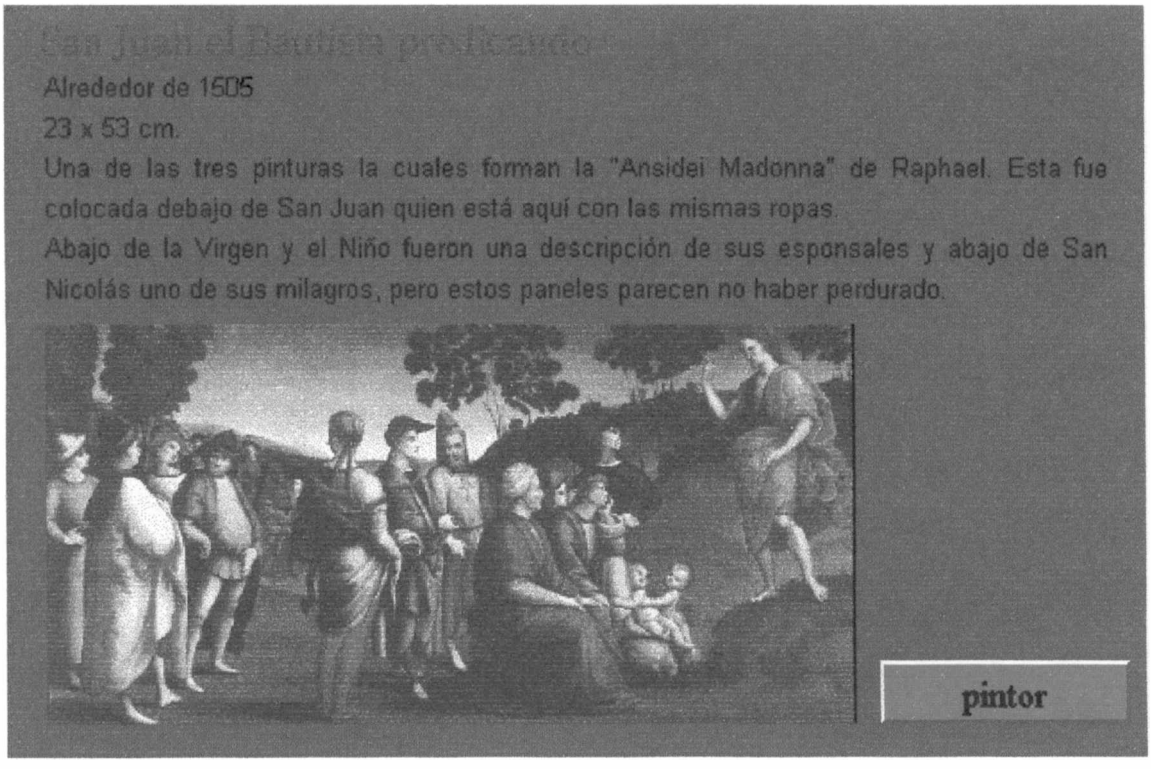


Figura 17. San Juan el Bautista predicando de la hiperbase *ejemplo detallado.*

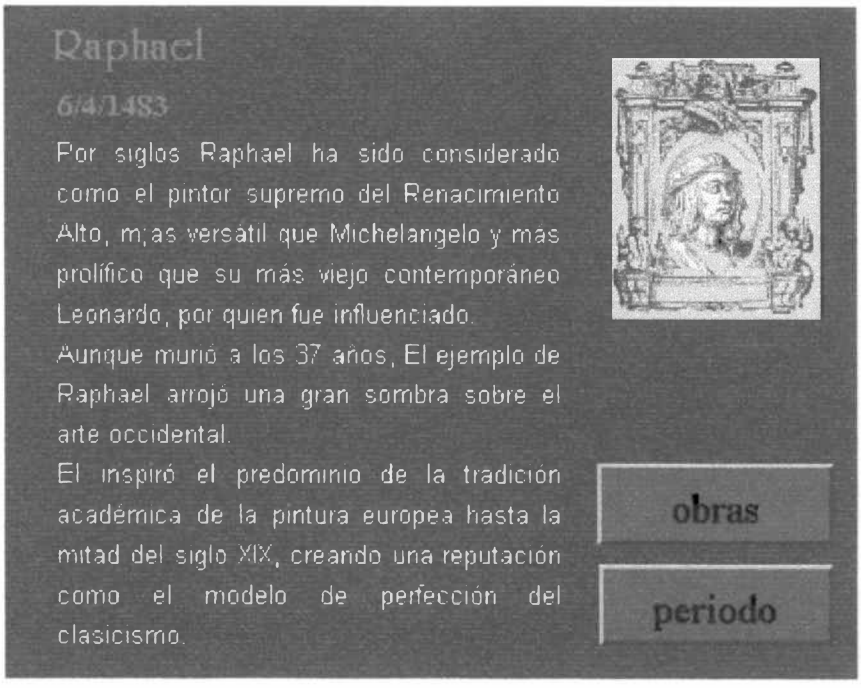


Figura 18. Raphael de la hiperbase *ejemplo detallado.*

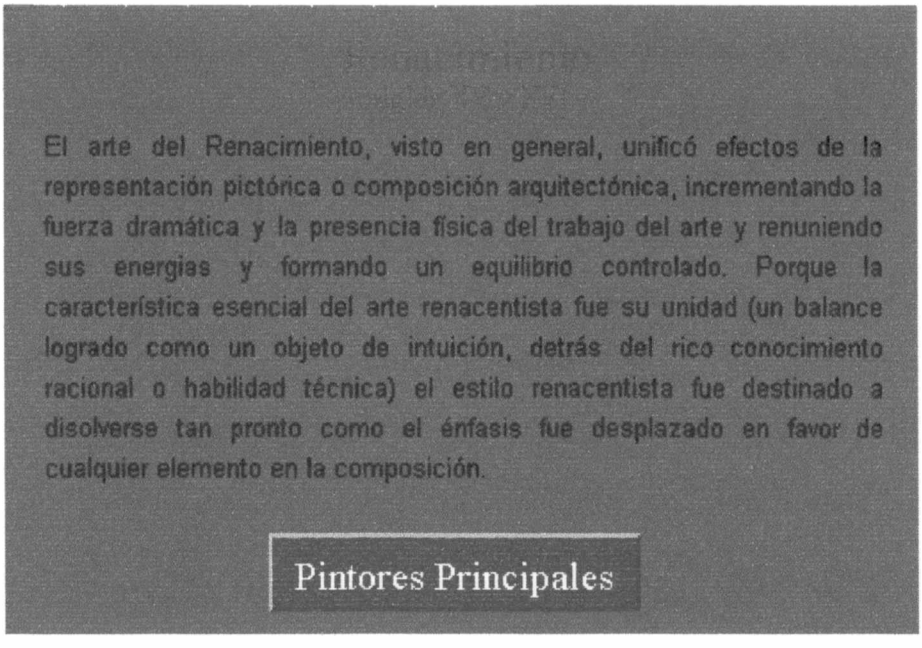


Figura 19. Período Renacimiento

Las figuras 17 y 18 muestran dos nodos de la vista navegacional *ejemplo detallado* en los que se puede observar que se presenta una mayor cantidad de infor-

mación tanto de los pintores como de las pinturas. En la figura 19 se ve un nodo Período que fue diseñado especialmente para esta vista navegacional.

5. CARACTERÍSTICAS DE LA HERRAMIENTA

En esta sección detallaremos cuáles son las características principales de nuestra herramienta, las cuales justificarían su uso sobre los demás sistemas de autoría existentes.

Una característica fundamental es que permite la conexión entre un modelo de objetos existente y una implementación particular. Esto se logra estableciendo relaciones entre las clases navegacionales y las clases conceptuales (no necesariamente la relación será uno a uno, ya que una clase navegacional puede derivarse de más de una clase conceptual, o la clase navegacional puede reflejar solo parte de la correspondiente clase conceptual), las relaciones se establecen a nivel de atributos, es decir un atributo de la clase navegacional se relaciona con una variable de instancia de la clase conceptual (esto es lo que permite la derivación desde distintas clases o el relevamiento parcial). Hasta aquí, la relación está establecida en un alto nivel; luego, durante el desarrollo de la hiperbase, se establecerá la relación con una instancia en particular de la/s clase/s conceptual/es correspondientes al nodo que se está instanciando. Además, la hiperbase creada sobre un modelo de objetos tiene la característica especial de que si el modelo de objetos es modificado los cambios quedarán automáticamente reflejados en ella.

Otra característica muy interesante es la generación automática de índices. Cuando se define una estructura de acceso global basta con señalar cuál es el nodo destino, cuál el atributo selector y qué predicado se quiere aplicar. Luego el índice se generará automática y dinámicamente según los nodos de la hiperbase que corresponden al tipo del nodo destino, que en el momento de ser invocado, cumplan el predicado.

Otra ventaja es el uso de links tipados, los cuales incluyen en su definición el origen y el destino. Esto aporta importantes ventajas como: mayor comprensión

de la estructura de navegación, facilidades para la interconexión de los nodos (cada vez que se establece un link hay un chequeo de tipo subyacente de que el tipo de los nodos origen y destino sean correctos).

Una ventaja con respecto a Toolbook es la posibilidad de generar nodos con ventanas de tamaños diferentes; permitiendo al autor una mayor flexibilidad en el diseño de la interface.

6. IMPLEMENTACIÓN

6.1. Cuadro General

La herramienta fue desarrollada en Smalltalk-80 y se usó VisualWorks 1.0 para el diseño de la interface.

El sistema está organizado en editores independientes encargados cada uno de una parte principal del mismo: la clase *EditorNodos* tiene como responsabilidad la definición de los nodos navegacionales; la clase *EditorLinks* es la encargada de los links navegacionales; la clase *EditorAcceso* controla las estructuras de acceso; la clase *EditorSubsistemas* está a cargo de la definición de los subsistemas; la clase *EditorGrafos* permite dibujar los esquemas de las hiperbases a desarrollar; la clase *EditorHiperbase* se responsabiliza de la construcción de las hiperbases y la clase *Navegacion* permite las facilidades de navegación a través de una hiperbase. Instancias de estas clases están agrupadas en la clase *Principal* como valores de variables de instancia, y está última se encarga del acceso a los editores mediante una botonera. También esta clase tiene las instancias de *LinePalette* (Paleta de línea) y *ColorPalette* (Paleta de colores); y la colección de los proyectos ya desarrollados.

La clase ***EditorNodos*** tiene una instancia de la clase ***EditorPerspectiva***, encargada de la ventana donde se diseñan las perspectivas, y una instancia de la clase ***RelacionarDialog***, que permite la conexión del nodo navegacional con el modelo conceptual. También manipula la colección de nodos navegacionales definidos para el proyecto actual, la información de cada uno de ellos es guardada en una instancia de la clase ***NodoRecord***, y es la siguiente: *nombre*, nombre del nodo; *superclase*, superclase del nodo; *atributos*, colección de instancias de ***Atributo*** (cuyas variables de instancia son *nombre* y *tipo*); *perspectivas*, colección de instancias de ***PerspectivaRecord*** (con las siguientes variables de instancia: *nom-*

bre, nombre de la perspectiva; *backColor*, color de fondo; *tamaño*, tamaño de la ventana; y *view*, colección de componentes de la perspectiva, por ejemplo, líneas, cuadrados, etc.); y *relaciones*, colección de instancias de ***RelacRecord*** (las variables de instancia de éste son *claseR*, clase del modelo conceptual; *atributo*, atributo del nodo que se relaciona; *varInst*, variable de instancia relacionada; *selector*, lista de métodos para acceder al valor de la variable de instancia; y *grupo*, grupo al que pertenece la relación).

La clase ***EditorLinks*** también tiene una instancia de la clase ***EditorPerspectiva***, para el caso de links con nodo intermedio. Además maneja la colección de links navegacionales del proyecto actual, que son instancias de ***LinkRecord*** cuyas variables de instancia son *nombre*, nombre del link; *superclase*, superclase del link; *cardi*, cardinalidad del link; *origen*, nombre del tipo de nodo navegacional origen; *destino*, nombre del tipo de nodo navegacional destino; *atributos*, si los tiene, colección de instancias de la clase ***Atributo***; y *perspectiva*, si la tiene, una instancia de ***PerspectivaRecord***.

La clase ***EditorAcceso*** contiene las estructuras de acceso definidas en el actual proyecto, éstas son instancias de la clase ***AccesoRecord*** definida con estas variables de instancia: *nombre*, nombre de la estructura de acceso; *destino*, el tipo de nodo navegacional destino; *selector*, atributo identificador; y *predicado*, predicado a aplicar a las instancias de la clase destino. Y usa la clase ***PredicadoEditor*** para crear los predicados de las estructuras de acceso.

Los subsistemas definidos en el proyecto están a cargo de la clase ***EditorSubsistemas*** y son instancias de la clase ***SubsiRecord*** cuyas variables de instancia son *nombre*, nombre del subsistema; *clases*, colección de clases que pertenecen al subsistema; y *entrys*, colección de puntos de entrada.

EditorHiperbase es la clase responsable de manejar las hiperbases, colección de instancias de ***HiperRecord*** con las siguientes variables de instancia: *nombre*, nombre de la hiperbase; *nodos*, colección de instancias de la clase ***InstanciaRecord*** (subclase de ***PerspectivaRecord*** y agrega las variables de instancia *tipo*, nombre del nodo navegacional del cual es instancia; y *subsistema*, subsistema al que pertenece); *inicial*, nodo inicial cuando se abre la hiperbase; *flags*, conjunto de banderas que indican cuáles botones estarán presentes durante la navegación; *versiones*, conjunto de distintos recorridos que se han iniciado en la hiperbase, la colección es de instancias de la clase ***VersionRecord*** (las variables de instancia de éste son: *nombre*, nombre de la versión; *historia*, colección de nodos recorridos; y *actual*, nodo en el cuál se abandono la navegación); *tour*, recorrido que el autor ha diseñado para ofrecer al usuario final; y *globales*, colección de instancias de estructuras de acceso definidas en la hiperbase, esta colección es de instancias de ***GlobalRecord*** (*nombre*, nombre de la instancia de la estructura de acceso; *acceso*, estructura de acceso del cual es instancia; *text*, estilo del texto al abrir la estructura de acceso durante la navegación; y *look*, colores del texto y del fondo).

La clase ***Navegación*** tiene una instancia de ***LectorView***, (ver sección 6.1.1.).

La clase ***EditorGrafos*** tiene una instancia de ***GrafoView*** (ver sección 6.1.1.) y otra de ***GrafoPalette***, esta última contiene la barra de herramientas para poder dibujar un esquema. El conjunto de esquemas dibujados está almacenado en la variable de instancia *grafos*, que es una colección de ***PerspectivaRecord***.

La clase ***EditorPerspectiva*** es usada para definir las perspectivas de los nodos o la perspectiva de un link. Tiene una instancia de ***ToolPalette***, paleta de herramientas para diseñar la perspectiva y una instancia de ***EditorView***, (ver sección 6.1.1.).

Para los editores que permiten dibujar objetos en la ventana, editor de perspectivas y editor de hiperbase, tuvimos la necesidad de definir nuestro propios componentes gráficos. Cada uno de éstos, además de automostrarse, controla la modificaciones que se pueden hacer sobre el mismo, tales como, flip horizontal, flip vertical, transparente, etc. La mayoría de los mismos son subclases de ***Componente*** (subclase de SimpleComponent), excepto ***Boton*** que es subclase de ActionButtonView y ***Texto*** (y sus subclases) que es subclase de TextEditorView. La jerarquía puede verse en la figura 20.

```
ActionButtonView
        Boton (origen, corner, ancho, destino, atributo, objeto, linkeado)
---
TextEditorView (state)
        Texto (origen, corner, ancho, modelo)
                Field (atributo, objeto)
                        FieldString (linkeado)
                                FieldFecha
                                FieldNumero
                NodoGrafico (links, subsistema, puntos)
                        AccesoGrafico
---
SimpleComponent (state)
        Componente (origen, corner, ancho)
                Linea
                        LinkGrafico( fuente, destino)
                                EsUn
                                Relacion (nombre)
                                        Reflexivo
                Rectangulo
                Ellipse
                Curva (stang, swang)
                Multilinea (vertices, relativo)
                        Poligono
                Imagen (imagen)
                        ImagenAtrib (atributo, objeto, linkeado)
```

Figura 20. Jerarquía de componentes gráficos

6.1.1. Arquitectura MVC

El método de descomposición de una aplicación Smalltalk es conocido como arquitectura Model-View-Controller (MVC).

Una aplicación se divide en dos partes:

- el modelo de información, el cual maneja el almacenamiento de los datos y el procesamiento.
- Interface para el usuario, la cual maneja las entradas y salidas.

La razón de esta división es que un componente dado de la interface de usuario es más probable de ser usado en otra aplicación si éste no depende de los detalles de implementación de un dominio en particular. También, se puede desear sustituir una interface para el usuario diferente para un mismo modelo, por ejemplo, una interface para los usuarios novatos que oculta facilidades para usuarios expertos. MVC encapsula estas dos partes funcionales de una aplicación tanto como sea posible.

Primero se desarrolla el **modelo**, al menos de una manera rudimentaria, porque la interface de usuario necesita de donde obtener la información. La interface para el usuario está subdividida en un componente para mostrar la salida (conocido como una **view**) y otro componente que permite al usuario controlar o interactuar con la aplicación (un **controller**). El controller maneja la entrada del usuario tales como selección en un menú o actividad en el teclado. En términos tradicionales, se puede pensar a la view como la salida de una aplicación, el controller como la entrada de una aplicación, y al modelo como el procesamiento de la aplicación.

El motivo para separar la entrada de la salida es el mismo que el que separa la interface para el usuario del modelo de información, permitiendo mezclar módulos de entrada y salida. Por ejemplo, un usuario novato podría ver la misma in-

formación (view) que un usuario experto, pero el menú de comandos podría ser simplificado sustituyendo el controller.

Las views y los controllers están íntimamente relacionados, cada view tiene solamente un controller (o ninguno) y cada controller trabaja con una simple view. Es más, la view es capaz de crear su propio controller. El modelo se comunica solamente con la view.

En nuestro proyecto cada editor es manejado como una aplicación independiente, con su propio modelo, view y controller. En algunos casos la view y el controller provistos por Smalltalk son suficientes para la interacción con el usuario, pero en otros casos necesitamos crear nuestras propias views y nuestros propios controllers. Para estos casos se crearon las clases ***ViewGeneral*** y ***ControllerGeneral*** que reúnen el comportamiento y la estructura común de las distintas views y controllers necesarios para cada uno. La jerarquía de clases es la siguiente, ViewGeneral es subclase de CompositeView (esta clase fue elegida porque nos provee de un *controller* y un *model* y además permite que en la view pudiese haber múltiples *components*); sus subclases son EditorView, HiperView y LectorView, éstas se diferencian principalmente en cuál es su clase de controller. GrafoView es subclase de CompositeView y no de ViewGeneral, ya que difiere de las demás views (principalmente los componentes gráficos son distintos). Por otro lado, ControllerGeneral es subclase de ModalController y sus subclases son EditorController, HiperController, LectorController y GrafoController (figura 21).

CompositeView
    ViewGeneral
        EditorView
        HiperView

ModalController
    ControllerGeneral
        EditorController
        HiperController

LectorView LectorController

GrafoView GrafoController

Figura 21. Jerarquía de views y controllers

Los casos en que creamos nuestra propia view y nuestro propio controller se ven en la siguiente tabla.

	Model	View	Controller
Editor de esquemas	EditorGrafos	GrafoView	GrafoController
Editor de perspectivas	EditorPerspectiva	EditorView	EditorController
Editor de hiperbase	EditorHiperbase	HiperView	HiperController
Herramienta de navegación	Navegacion	LectorView	LectorController

Fue necesario realizar estos cambios debido a que necesitábamos controlar los eventos del mouse dentro de la ventana según qué elemento de la paleta de herramientas de cada editor estuviese seleccionado, tomando las acciones pertinentes a cada caso. Por otro lado al tener nuestros propios componentes gráficos también necesitábamos controlar la visualización de los mismos.

6.2. Problemas y Soluciones

Los principales problemas que tuvimos al realizar al proyecto fueron los siguientes:

- En el editor de nodos cuando un atributo ya está reflejado en alguna perspectiva, si luego era borrado o modificado la perspectiva quedaba inconsistente con la definición del nodo navegacional; para solucionar este problema decidimos no permitir modificar un atributo que estuviera en alguna perspectiva. Si se de-

sea hacerlo, antes se deberá borrar el atributo de todas las perspectivas en las que esté presente y si borra un atributo también se borrará de todas las perspectivas en las que esté presente.

- Para relacionar una clase navegacional con una clase conceptual tuvimos que obligar al usuario a que al menos tuviese una instancia de la clase conceptual ya creada, esto es porque era necesario verificar la correctitud de los métodos de acceso a las variables de instancia de la clase conceptual. Incluso si se está relacionando un atributo de tipo Anchor, la clase conceptual destino también deberá tener al menos una instancia para poder elegir un método identificador para el destino.

- Cuando se define un atributo de tipo Anchor, pedimos al usuario que ingrese el tipo de link de éste, en principio no es necesario que ya esté definido en el proyecto (mediante el editor de links), pero sí deberá definirlo antes de relacionar el atributo con el modelo conceptual, porque es necesario conocer la cardinalidad del link para poder relacionarlo correctamente (si el link tiene cardinalidad 1 a N en el modelo conceptual esperamos encontrar una Collection de los objetos destinos, si no estuviese explicitado podría pensar que la cardinalidad es 1 a 1 y el destino es la collection y no sus elementos).

- Dado que nuestra herramienta es para construir aplicaciones hipermedia, debíamos facilitar la incorporación de gráficos a un nodo. Para esto tuvimos que investigar sobre la estructura de los archivos BMP y PCX; el problema principal que tuvimos fue en poder ajustar la paleta de colores del archivo con la de Smalltalk.

- VisualWorks nos ofrecía sólo un conjunto limitado de tipos de fuentes (solamente 5 tipos). Tuvimos problemas al intentar agregar los tipos de fuente provistos por Windows, la fuente quedaba instalada pero al salir de Smalltalk

y volver a entrar ya no estaba más. Para solucionar esto, la clase *Principal* tiene una colección de las fuentes usadas en el desarrollo de la herramienta y las agregadas mas tarde por cualquier usuario, y cuando arranca VisualWorks reinstala estas fuentes para que estén disponibles.

- Otro problema que se nos presentó fue que permitíamos borrar los objetos relacionados con atributos (fields, botones, etc.) en la etapa de autoría, por lo tanto, un usuario podía transformar un nodo navegacional borrando atributos, perdiéndose de esta manera el concepto que representa la definición de dicho nodo navegacional. Para evitar esto resolvimos no permitir el borrado de objetos relacionados con atributos durante la edición de hiperbases, y sólo permitimos modificar el aspecto del objeto (color, fuente, tamaño etc.).

- Varias problemas surgieron al implementar el linkeo automático de nodos de la hiperbase. El objeto Anchor referencia a la instancia del modelo conceptual correspondiente al destino del link, el problema que tuvimos era cómo determinar cuál nodo de la hiperbase correspondía a esa instancia, debido a que un nodo puede ser una vista parcial o una combinación de objetos del modelo conceptual. Lo que resolvimos fue: de entre todos los nodos posibles de ser el destino elegimos el que contiene más atributos del objeto del modelo conceptual (por lo tanto el destino puede variar a medida que la hiperbase se agranda). Otra resolución que tomamos fue no permitir al usuario modificar los links establecidos automáticamente, tomamos esta decisión para evitar que el usuario viole el modelo conceptual; aunque permitimos instanciar un nodo sin relacionarlo con el modelo conceptual, caso en el cual el usuario linkea manualmente los nodos. Otros inconvenientes se presentaron para el caso de links con cardinalidad 1 a N, ya que además de determinar las instancias, también teníamos que generar el pop-up que aparece al cliquear el botón.

- En principio la herramienta no facilitaba una visualización global del proyecto que se estaba realizando, es decir, los nodos solo se veían en el editor de nodos, los links en el editor de Links, etc.. Para resolver esto diseñamos el editor de esquemas que permite ver la relación entre nodos, links, estructuras de acceso, etc., además de poder acceder a los editores correspondientes cliqueando sobre el objeto deseado, generando automáticamente algunos datos, como por ejemplo, la superclase y los atributos anchor para un nodo; o el origen y el destino para un link. Como el esquema dibujado solo es una presentación visual del proyecto, no se verifica que posteriores modificaciones en la definición de las clases navegacionales se correspondan con el esquema dibujado previamente.

- En principio el lector perdía el contexto de navegación cada vez que salía, es decir, las marcas de nodos visitados, la historia del recorrido y el nodo en el cual abandonó la navegación. Por lo tanto, debía volver a recorrer la hiperbase para llegar a la posición donde había quedado la última vez. Problema aún mas severo en el caso de grandes hiperbases. Decidimos que el lector pudiese guardar la información al salir, entonces el lector podrá retomar la navegación tal cual la había dejado. Como puede haber más de un lector para la misma hiperbase decidimos que cada lector pudiese grabar la información eligiendo un nombre.

- Otro problema que se nos presentó es que teníamos todos las clases navegacionales de distintas aplicaciones presentes durante el desarrollo de alguna aplicación en particular, complicándose la tarea de diseño para el autor al tener que seleccionar las clases que le interesan entre todas las definidas en la herramienta. Para resolver esto decidimos usar el concepto de *proyecto*, permitiendo al autor definir el contexto para cada aplicación que desarrolla, e inclu-

so agregamos la función *importar* en cada editor para permitir el acceso a clases navegacionales ya definidas para otro proyecto.

- Cuando el autor relaciona el modelo conceptual con el diseño navegacional debe elegir las clases Smalltalk que pertenecen al primero. El problema que se nos presentaba es que debía seleccionar estas clases entre **todas** las clases de Smalltalk, tarea que resultaba bastante engorrosa para el autor. Por lo tanto, decidimos que cada vez que se empieza un nuevo proyecto el autor indique la *categoría*[1] en la cual previamente agrupó las clases conceptuales y así achicar de manera considerable el espacio de búsqueda.

[1] Categoría: Smalltalk permite definir categoría de clases, las cuales agrupan clases que tienen una funcionalidad en común.

7. POSIBLES EXTENSIONES

- Una extensión posible es agregar un tipo de atributo *Sonido*. Al estar asociado a un atributo se activaría mediante botones.

- Permitir manejadores tales como: de entrada a un nodo, salida de un nodo, entrada y salida del mouse a un objeto, etc. Estos manejadores necesitarían de un lenguaje que podría ser el mismo Smalltalk o un lenguaje tipo *Script de Toolbook* mas amigable a los usuarios que Smalltalk.

- Otra extensión es que la herramienta permita el desarrollo del modelo conceptual directamente. El usuario dibujaría el esquema conceptual y automáticamente (o no) se generarán en Smalltalk las clases y relaciones definidas con sus atributos.

- Que la importación de imágenes incluya una mayor cantidad de formatos de gráficos (GIF, RLE, WMF etc.).

- Por el momento los índices al ser invocados abren un menú *pop-up*, sería interesante que también puedan ser un nodo más de la hiperbase, donde el autor pudiese diseñar con cierta libertad la apariencia del nodo índice.

- Incorporar el atributo gráfico como posible selector de una estructura de acceso.

- Mejorar el predicado de las estructuras de acceso con expresiones más complejas.

- Los Links con cardinalidad 1 a N no pueden tener atributos ya que más tarde en el Editor de Hiperbase no se podrán definir nodos intermedios para cada destino en particular, sería importante que sí los tuviesen.

- En un modelo de objetos para modelar un link que tiene atributos se debe crear una clase con los atributos y el destino final, por lo tanto nuestra herramienta no transforma esta clase a un link con nodo intermedio, sino que es un nodo navegacional más de la hiperbase.

- Los botones *back, inicio, next y previous* están siempre fijos en una barra en la parte superior de la ventana; si bien puede elegirse que la barra no se vea, sería interesante que el autor pudiese ubicarlos en cualquier lugar de la ventana que él desee, proveyendo mayor libertad para el diseño de la interface.

- Proveer diferentes estilos para los botones para evitar la monotonía de tener solo botones *push*, tales como: sin borde, recuadrados, sombreados, redondeados, etc..

8. CONCLUSIONES

Un aspecto fundamental es haber logrado la conexión con un modelo conceptual desarrollado en Smalltalk lo que permite una rápida conversión de un sistema orientado a objetos a multimedial, al poder aprovechar todas las instancias creadas para el primero. Además, una hiperbase creada a partir de un modelo conceptual en Smalltalk se actualizará dinámicamente al producirse un cambio en alguna de las instancias de este modelo.

La generación automática de índices permite la actualización dinámica de las estructuras de acceso globales cuando nuevos nodos se van agregando a la hiperbase, es decir, las estructuras de acceso no necesitan ser modificadas al producirse cambios en la hiperbase. Esta ventaja agiliza sustancialmente la etapa de mantenimiento.

Si queremos derivar varias aplicaciones de un mismo modelo conceptual, *vistas navegacionales* [Schwabe95], el sistema permite reusar los nodos navegacionales y tipos de links definidos para otra aplicación, facilitando ampliamente la tarea. Esto se puede hacer a través de distintas perspectivas para el mismo nodo, o bien definiendo un nuevo proyecto e importando los nodos y links que queremos reusar.

9. REFERENCIAS

[Garzotto91] F. Garzotto, P. Paolini and D. Schwabe: "HDM - A Model for the Design of Hypertext Applications", Proceedings of Hypertext'91, ACM Press. Pp. 313

[Garzotto93] F. Garzotto, D. Schwabe, P. Paolini: "HDM - A Model Based Approach to Hypermedia Application Design", ACM Transaction on Information Systems, Vol. 11 N° 1, Enero 1993, pp. 1-26.

[Isakowitz95] T. Isakowitz; E. Stohr; P. Balasubramaniam, "RMM, A methodology for structured hypermedia design", Comm ACM, August 1995, pp 34-48.

[Lange93] D. Lange "Developing Hypermedia Information Systems: An Object-Oriented Approach". 1993

[Lange94] D. Lange "An Object-Oriented Design Method for Hypermedia Information Systems". 1994.

[Rossi95a] G. Rossi; D. Schwabe; C.J.P. de Lucena; D.D. Cowan, "An Object-Oriented Model for Designing the Human-Computer Interface of Hypermedia Applications", Proc. of the International Workshop on Hypermedia Design (IWHD'95), Springer Verlag Workshops in Computing Series, forthcoming.

[Rossi95b] G. Rossi; A. Garrido; S. Carvalho, "Object-Oriented Patterns for Hyprmedia Applications", Proceedings of Patterns Languages of Programs (PLOP'95), forthcoming.

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

[Schwabe93] D. Schwabe y G. Rossi: "Introdução aos Sistemas e à Autoria Hipermídia". EBAI '93, Rio Tercero, Argentina.

[Schwabe94a] D. Schwabe, G. Rossi, "From Domain Models to Hypermedia Applications. An Object-Oriented Approach", International Workshop on Methodologies for Designing and developing Hypermedia Applications, Edinburgh, September 1994.

[Schwabe94b] D. Schwabe and G. Rossi: "OOHDM. An Object-Oriented Hypermedia Design Model". COMM of the ACM, August 1995.

[Schwabe95] D. Schwabe and G. Rossi: "Defining Hypermedia Applications as Navigational Views of Hyperbases". Int. Conf. on System Science, Hawaii, 1995.

APÉNDICE A. MANUAL DE USUARIO

1. Menú Principal

El menú principal (figura 1) permite el acceso a todas las funciones de la herramienta.

Consta de:

- Botón *Editor de Esquemas*. Abre el editor de esquemas (ver sección 2).
- Botón *Editor de Nodos*. Abre el editor de nodos (ver sección 3).
- Botón *Editor de Links*. Abre el editor de links (ver sección 4).
- Botón *Editor de Subsistemas*. Abre el editor de subsistemas (ver sección 5).
- Botón *Editor de Estructuras de Acceso*. Abre el editor de estructuras de acceso (ver sección 6).
- Botón *Editor de Hiperbase*. Abre el editor de hiperbase (ver sección 7).
- Botón *Navegación*. Abre un cuadro de diálogo para seleccionar la hiperbase que se quiere navegar, y también se puede seleccionar el lector si es que ya fue recorrida anteriormente (ver sección 8).
- Botón *Salir*. Permite salir de la herramienta.

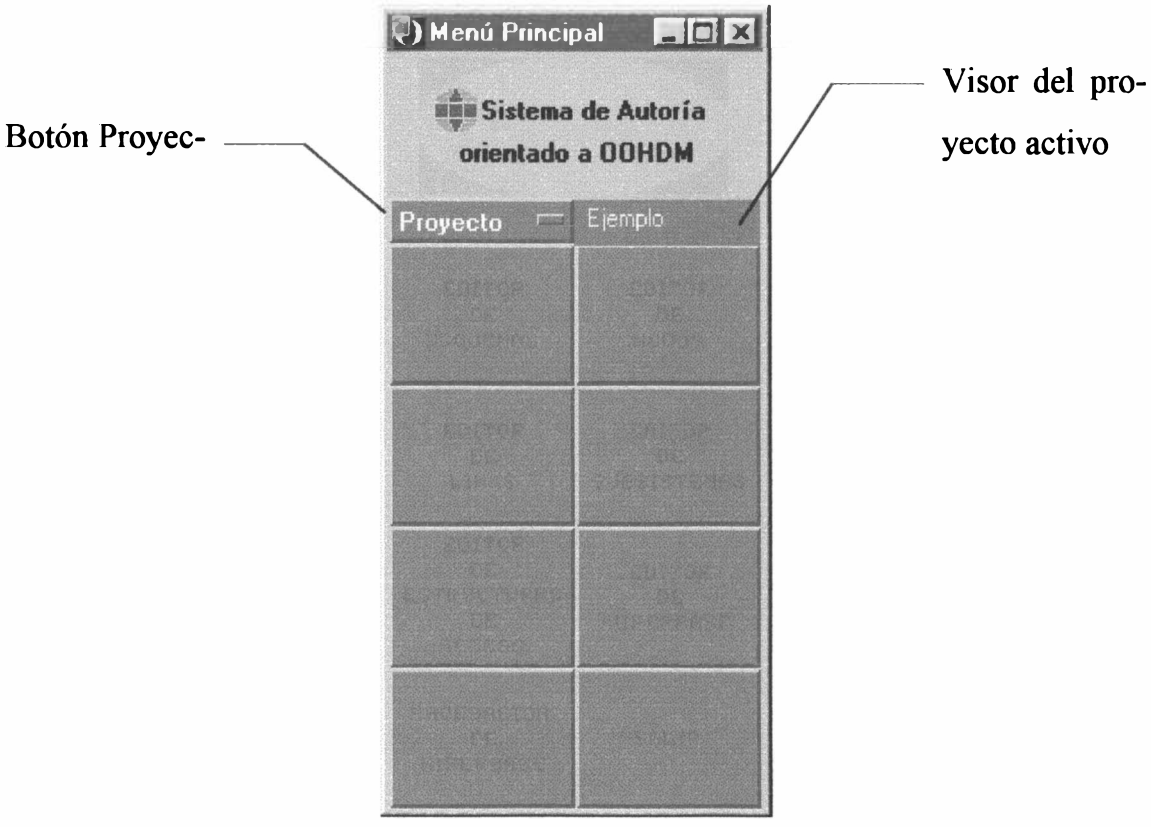


Figura 1. Menú Principal.

- Botón *proyecto*: al presionarlo se descuelga un menú con las siguientes opciones: nuevo, abrir y borrar. A su derecha hay un campo que indica cuál es el proyecto actual. La función *nuevo* permite crear un nuevo proyecto, se abre un cuadro de diálogo donde se pregunta por el nombre y por la categoría donde las clases del modelo conceptual se encuentran agrupadas (si es que existe). *Abrir* permite pasar a un proyecto ya existente, y *borrar* elimina un proyecto. CUIDADO: Los proyectos borrados no podrán ser recuperados.

2. Editor de Esquemas

El editor de esquemas sirve para representar gráficamente el esquema navegacional. El objetivo de este editor es que el usuario tenga una visión del esquema de la aplicación a desarrollar, pudiendo acceder a los editores a través de éste, simplificándose la tarea de diseño de los nodos navegacionales, links, etc..

Este editor tiene el menú *Archivo* con las siguientes opciones: *Nuevo*, *Abrir*, *Guardar*, *Guardar como*, *Borrar* y *Salir*.

Al invocarse se abre una ventana principal donde se dibujará el gráfico y una paleta de herramientas. La ventana principal tiene el clásico menú *Archivo* con las funciones: nuevo, abrir, guardar, guardar como, borrar y salir.

La paleta (figura 2) tiene los siguientes botones:

- *Seleccionar*: Permite seleccionar los objetos dibujados cliqueando sobre los mismos; manteniendo la tecla SHIFT apretada se podrá seleccionar mas de uno (solo se pueden multi-seleccionar los nodos navegacionales).
- *Nodo*: Presionando este botón se podrán agregar nodos al esquema cliqueando sobre el lugar donde desea ubicarlos. Luego de soltar el botón izquierdo del mouse, aparecerá un cuadro de diálogo preguntando por el nombre del nodo navegacional.
- *Link*: Este botón permite establecer un link entre dos nodos navegacionales. Mientras esté habilitado aparece un 'help' indicando que primero se cliquea sobre el nodo origen y luego sobre el destino, también aquí se pregunta por el nombre. La herramienta permite también dibujar links reflexivos.
- *Es-Un*: Permite establecer una jerarquía entre los nodos navegacionales. Para poder establecerla el nodo superclase debe estar, gráficamente, so-

bre la subclase; hay que tener en cuenta que primero se cliquea sobre la superclase y luego sobre la subclase.

- *Subsistema*: Este botón funciona distinto a los demás. Primero se seleccionan los nodos navegacionales que formarán parte de un mismo subsistema y recién entonces se presiona este botón. Esto provoca que los nodos navegacionales seleccionados queden pintados todos del mismo color distinguiéndose del resto de los subsistemas y nodos. Para agregar un nodo a un subsistema se deberán seleccionar todos los nodos del subsistema y el nuevo nodo a agregar. Para sacar un nodo de un subsistema se selecciona el nodo a eliminar y se presiona el botón. Si al apretar el botón hay también seleccionadas estructuras de acceso no serán tenidas en cuenta.
- *Estructuras de Acceso*: Para agregar una estructura de acceso primero se la ubica en el esquema y luego se cliquea sobre el nodo navegacional destino de la misma.
- *Editar*: Cuando se selecciona un objeto estando presionado este botón, se abre el editor correspondiente al objeto seleccionado. Los objetos dibujados en el esquema no existen como tales; por lo tanto el editor correspondiente se abre con un elemento nuevo que tiene solo los datos que pueden derivarse del dibujo; salvo que el elemento ya existiera, en este caso se abre este último, haciendo caso omiso a los datos del dibujo (No existe un chequeo de que el esquema se corresponda con los nodos, links, etc. existentes). Para editar un subsistema se debe cliquear sobre cualquier nodo perteneciente al subsistema deseado manteniendo la tecla SHIFT apretada.

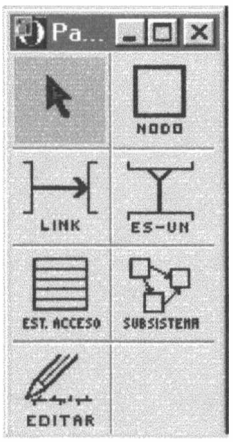


Figura 2. Paleta de herramientas del Editor de Esquemas.

Las estructuras de acceso y nodos pueden moverse libremente por la ventana, y sus links de entrada y salida se moverán con ellos. Los links no pueden moverse, para que el usuario no pueda dejar links desconectados sin origen y/o destino. Si un nodo es subclase de algún otro, y es movido encima de su superclase la línea que los une desaparecerá. Cualquier objeto seleccionado puede borrarse presionando la tecla DELETE, si se borra un nodo todos los links que salen y llegan además de las estructuras de acceso relacionadas con éste también se borrarán, esto es por lo mismo que antes, para mantener una coherencia en el dibujo.

A continuación en la figura 3 puede verse un ejemplo de un esquema.

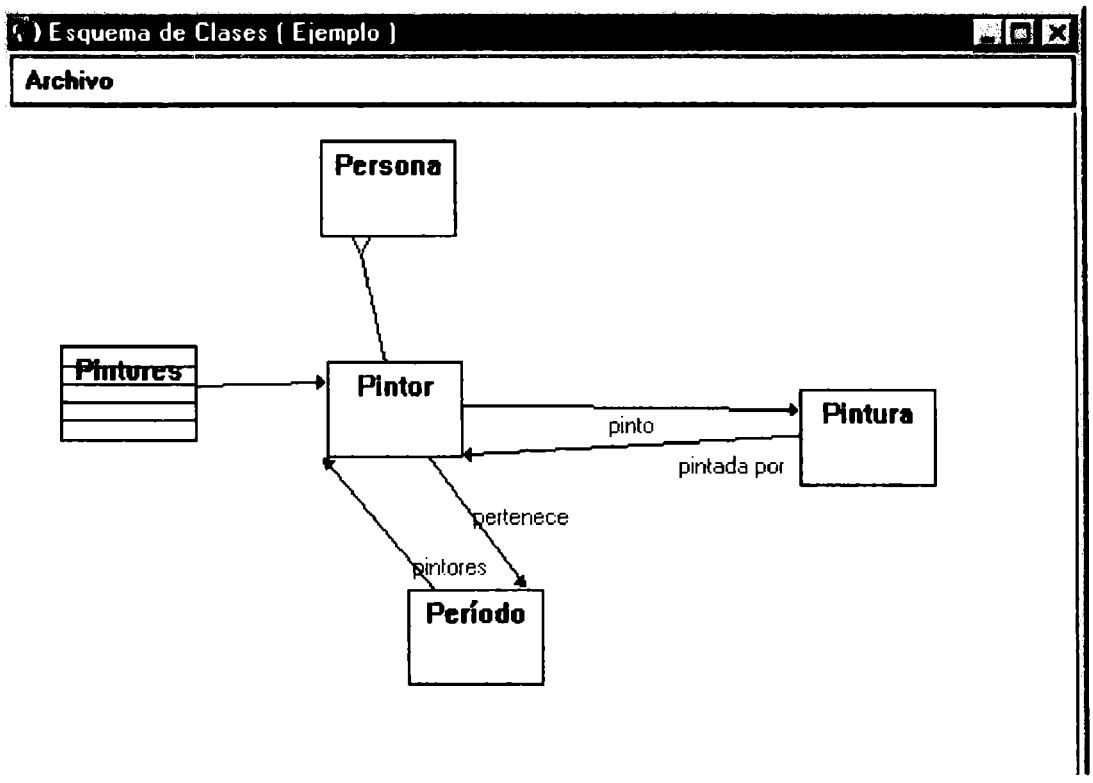


Figura 3. Editor de esquemas (ejemplo).

3. Editor de Nodos

En este editor se crean los nodos navegacionales, asignándoles un nombre, su correspondiente superclase, sus atributos y sus perspectivas (figura 4).

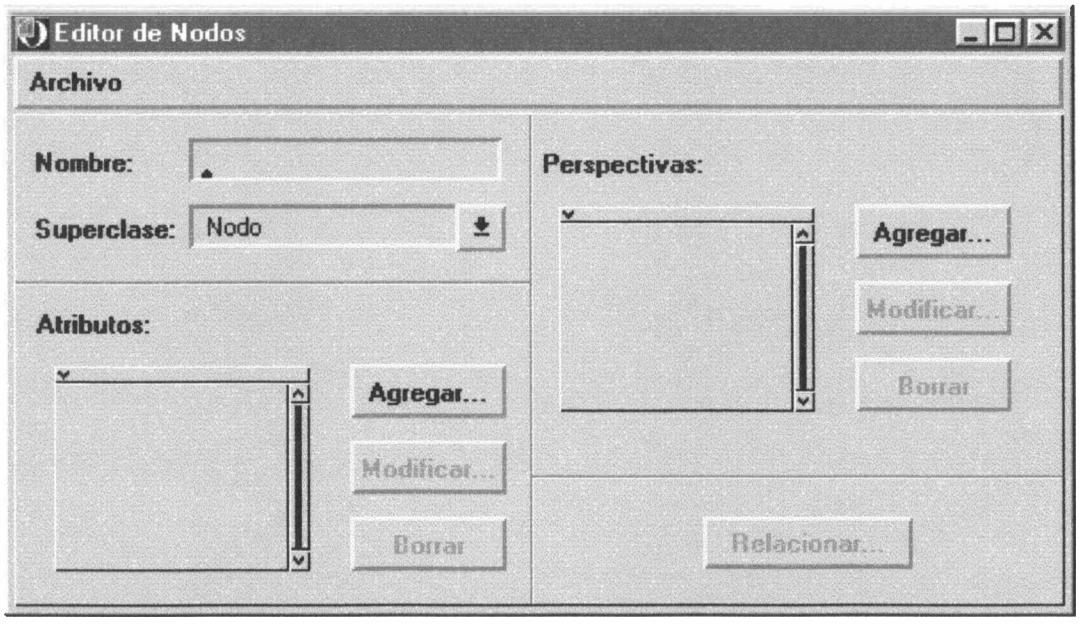


Figura 4. Editor de Nodos.

Este editor tiene el menú *Archivo* con las siguientes opciones: *Nuevo*, *Abrir*, *Importar* (permite importar nodos navegacionales de otros proyectos), *Salvar*, *Borrar* y *Salir*.

Para elegir la superclase se dispone de una lista de todas las clases de nodos ya existentes, única forma de completar el campo *superclase*. Existe un nodo navegacional abstracto, *Nodo*, que es la raíz de la jerarquía de nodos. Al seleccionar la superclase automáticamente aparecen en la lista de atributos los heredados según la jerarquía, separados de los propios del nodo por una línea; los atributos heredados no se pueden borrar ni modificar.

Junto a la lista de atributos están los botones *agregar*, *borrar* y *modificar*. Los botones *modificar* y *borrar* estarán deshabilitados si no hay un atributo seleccionado; el botón *borrar* borra el atributo seleccionado sin pedir confirmación, borrar un atributo implica que se borrará también de las perspectivas que lo contengan y la relación correspondiente al mismo; lo mismo para todas las perspectivas y relaciones de las subclases del nodo, obviamente, sino se graban los cambios

las subclases quedarán como estaban. El botón *agregar* abre un cuadro de diálogo para completar el nombre y el tipo correspondientes al atributo a agregar; en caso de seleccionar el tipo *anchor* este cuadro de diálogo se amplia para poder ingresar el tipo del link, hay una lista con los tipos de links ya existentes, aunque se puede escribir en el campo uno no definido aún (figura 5). El botón *modificar* abre el mismo cuadro de diálogo, pero con los campos indicando los datos del atributo seleccionado.

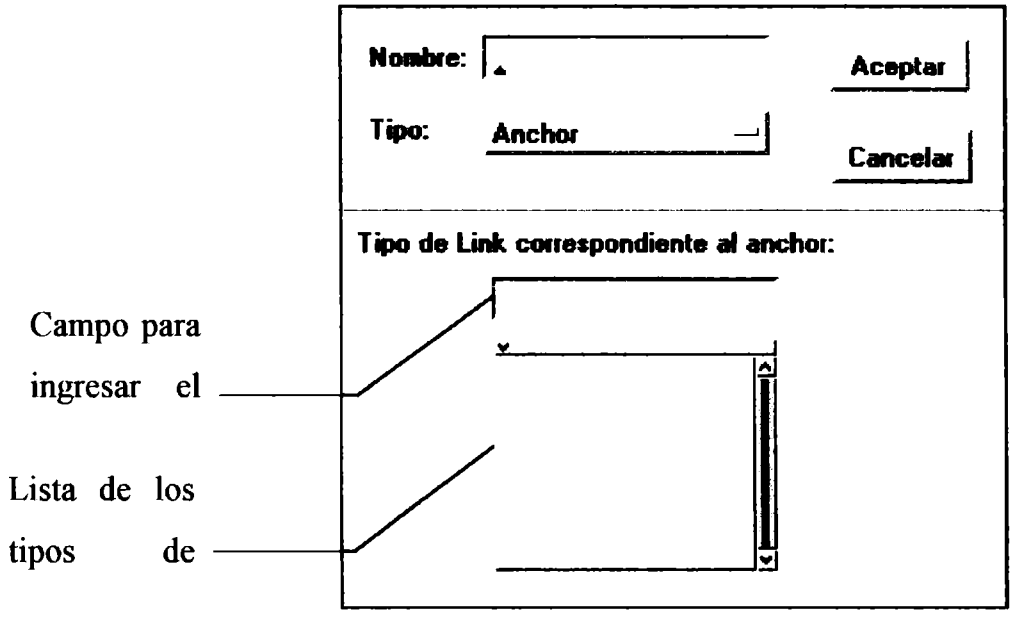


Figura 5. Cuadro de Diálogo para agregar atributos.

La lista de perspectivas también tiene a su derecha los botones *agregar, modificar* y *borrar*; igual que antes los botones *modificar* y *borrar* estarán deshabilitados si no hay una perspectiva seleccionada. Hay que tener mucho cuidado al borrar una perspectiva ya que no se pide confirmación (en caso de borrar una perspectiva sin querer, reabra el nodo sin salvar los cambios, solo sí ya estaba grabada antes). El botón *agregar* abrirá el editor de perspectivas (ver sección 3.1) pasando antes por un cuadro de diálogo que preguntará el nombre de la nueva pers-

pectiva; si al presionar el botón había una perspectiva de la lista seleccionada, se puede elegir entre crear una nueva a partir de la seleccionada o una nueva vacía; caso contrario sólo estará disponible la última opción. El botón *modificar* abre la misma ventana pero con la perspectiva seleccionada ya dibujada.

El botón *relacionar* abre una nueva ventana en la cual se establecerán las relaciones con las clases conceptuales del modelo (ver sección 3.2).

3.1 Editor de Perspectivas

El Editor de Perspectivas permite diseñar cómo será la interface del nodo navegacional. Al invocárselo se abre una ventana vacía (excepto que se abra una perspectiva para modificarla, o en el cuadro de diálogo para agregar se haya elegido *nuevo sobre seleccionado*); junto con una paleta de herramientas.

La ventana tiene una barra de menú con las siguientes opciones: *Archivo*, *Edición*, *Texto*, *Objeto*, *Dibujo*, *Ventana*. Dentro de *Archivo* están las opciones *salvar* y *salir*; esta última pregunta si se salvan los cambios. *Edición* tiene un submenú con las funciones *cortar*, *copiar* y *pegar* y también *seleccionar Todo*. El submenú de *Texto* tiene las siguientes opciones: *Fuente* abre un cuadro de diálogo para seleccionar el tipo, tamaño y estilo de la fuente, si hay objetos seleccionados la elección se aplica sobre éstos, sino lo que se setea es la fuente por defecto; *Párrafo* que tiene un submenú que permite alinear a izquierda, alinear a derecha, centrar o justificar el texto seleccionado, esta opción no está disponible para el texto de los botones; y las opciones de *negrita, cursiva* y *subrayado*. La opción *Objeto* maneja el nivel de profundidad de los objetos mediante las funciones *traer al frente*, *mandar atrás*, *traer adelante* y *mandar al fondo*. En *Dibujo* se encuentran las opciones *transparente* y *opaco*; además de *flip horizontal*, *flip vertical*, *rotar izquierda* y *rotar derecha*. También está *alinear* con un submenú con

las opciones *arriba, abajo, izquierda, derecha, centrado vertical y centrado horizontal*, esta opción es para aplicar sobre un grupo de objetos, el alineado es respecto al primer objeto seleccionado del grupo. Y por último, *Ventana* que abre la paleta de colores y la paleta de línea, los valores de las paletas se corresponden con el objeto seleccionado y para cambiar el color del background se debe cliquear en éste.

La paleta de herramientas cuenta con los siguientes botones (figura 6):

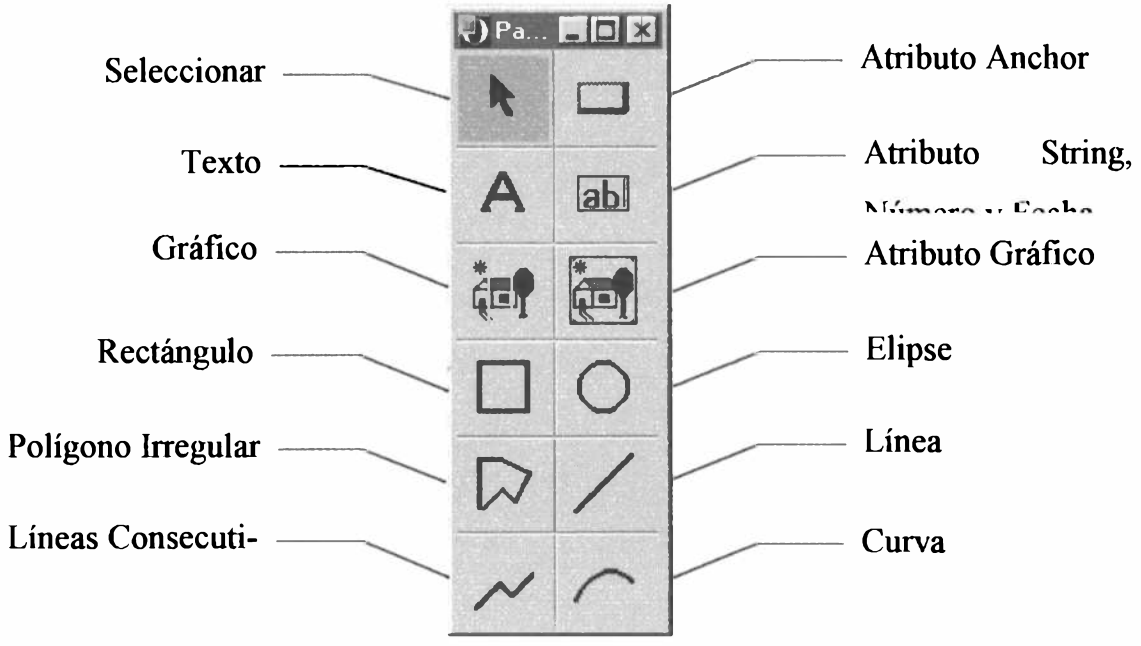


Figura 6. Paleta de Herramientas del Editor de Perspectivas

- *Seleccionar*: Permite seleccionar los objetos dibujados cliqueando sobre los mismos; manteniendo la tecla SHIFT apretada se podrá seleccionar mas de uno y también se pueden seleccionar varios objetos a la vez "dibujando" un rectángulo alrededor de ellos.

- *Texto*: Permite agregar texto a la perspectiva. Se dibuja el rectángulo que contendrá el texto, luego cuando está seleccionado se cliquea con el botón derecho dentro del mismo para editar el texto deseado. Las opciones para el tipo de letra, tamaño, alineación, etc. están disponibles en el menú *Texto*.
- *Rectángulo*: Para dibujar un rectángulo se cliquea en la ventana y manteniendo el botón apretado se lo extiende hasta la esquina opuesta.
- *Elipse*: En este caso lo que se dibuja es el rectángulo que contiene a la elipse deseada.
- *Línea*: Se cliquea en un extremo y manteniendo apretado se extiende hasta el otro extremo.
- *Curva*: Se dibuja igual que la línea.
- *Polígono Irregular*: Una vez seleccionada esta opción, se cliquea (el botón izquierdo) cada vez que se quiere insertar un vértice, y para terminar se cliquea el botón derecho donde estará el último vértice.
- *Líneas Consecutivas*: La funcionalidad es similar a la de los polígonos irregulares; es decir, se va cliqueando con el botón izquierdo cada vez que comienza una nueva línea y para terminar la última se cliquea el botón derecho.
- *Gráfico*: Se dibuja el rectángulo que va a contener el bitmap deseado, y cuando se suelta el botón se abrirá un "File Dialog" clásico (lista de unidades, lista de directorios, lista de archivos, etc.). Para el caso de unidades de discos intercambiables, el disco deberá ser colocado antes de dibujar el rectángulo.
- *Atributo Botón*: Se dibuja el botón de forma similar a un rectángulo; cuando se suelta el botón del mouse aparece un cuadro de diálogo para que se seleccione cuál es el atributo anchor correspondiente al botón dibujado.

- *Atributo String, Número y Fecha*: Se dibuja el campo y luego aparece el mismo cuadro de diálogo preguntando con qué atributo se relaciona al mismo.
- *Atributo Gráfico*: Igual a los dos anteriores, se dibuja el rectángulo que contendrá al gráfico y luego se selecciona cuál es el atributo de tipo gráfico correspondiente.

Todos los objetos pueden moverse libremente por la ventana y cualquier objeto seleccionado puede borrarse presionando la tecla DELETE (ésta funciona como cortar, es decir los objetos borrados van al portapapeles).

3.2 Relacionar

La parte superior de esta ventana contiene tres listas; la lista de atributos del nodo navegacional, la lista de clases conceptuales y la lista de variables de instancia de la clase conceptual seleccionada y el botón *relacionar* (figura 7). Para establecer una relación debe haber un elemento seleccionado de cada lista (caso contrario el botón *relacionar* estará deshabilitado); cuando se presiona este botón aparece un cuadro de diálogo para seleccionar cuál es el método que devuelve el valor de la variable de instancia. Teniendo en cuenta los siguientes casos:

- Si el atributo es String, Número, Fecha o Gráfico, se verifica que el método devuelva un valor atómico, sino no es así, se vuelve a pedir un método de la clase del objeto devuelto. Esto se repite hasta obtener un valor atómico.

- Para el caso de atributos *Anchor* su tipo de link deberá estas definido; además luego de verificar que el método elegido devuelve un objeto compuesto, se pide un método de la clase del objeto compuesto que devuelva un valor que

permita identificarlo. También, si el link es 1 a N se verifica que el método primeramente seleccionado devuelva una colección de objetos compuestos.

Nota: El modelo conceptual deberá estar completamente instanciado para que se puedan establecer las relaciones ya que el sistema efectúa un chequeo de las instancias para poder determinar si se están estableciendo bien las relaciones.

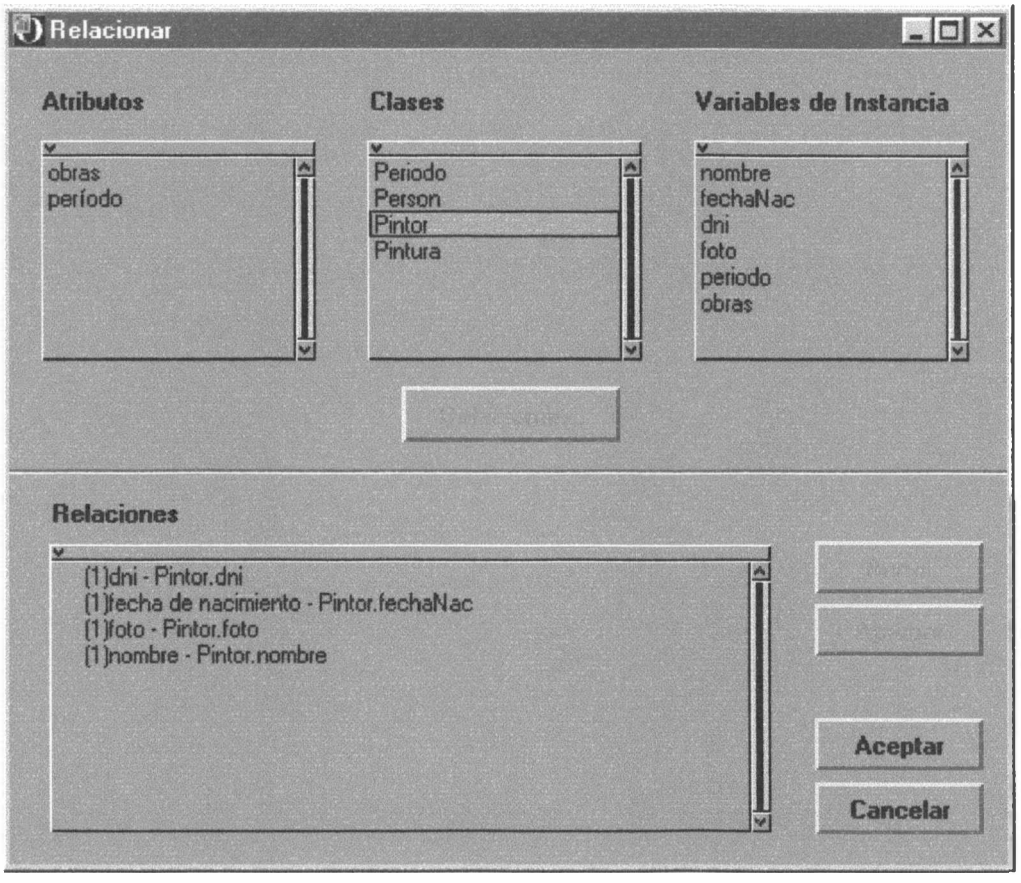


Figura 7. Cuadro de Diálogo Relacionar.

En la parte inferior se encuentra la lista de las relaciones ya establecidas y los botones *agrupar* y *borrar*. La lista de relaciones es multi-selección. El botón *agrupar* permite establecer conjuntos de atributos cuyos valores se tomarán de la misma instancia (generalmente todas las relaciones forman un único grupo), al presionarlo todas las relaciones selecciones formarán un nuevo grupo, entre pa-

réntesis delante de la relación aparecerá un número que distingue los distintos grupos. El botón *borrar* elimina todas las relaciones que estén seleccionadas.

4. Editor de Links

El Editor de Links (figura 8) permite definir los tipos de links del sistema, mediante el nombre, la superclase, el origen, el destino, los atributos y la cardinalidad .

Este editor tiene el menú *Archivo* con las siguientes opciones: *Nuevo*, *Abrir*, *Importar* (permite importar tipos de links desde otros proyectos), *Salvar*, *Borrar* y *Salir*.

Para elegir la superclase el mecanismo es similar al de editor de nodos, se dispone de una lista de todas las clases de links ya existentes. Existe un tipo de link abstracto, *Link*, que es la raíz de la jerarquía de links. Al seleccionar la superclase automáticamente aparecen en la lista de atributos los heredados según la jerarquía, separados de los propios del link por una línea; los atributos heredados no se pueden borrar ni modificar.

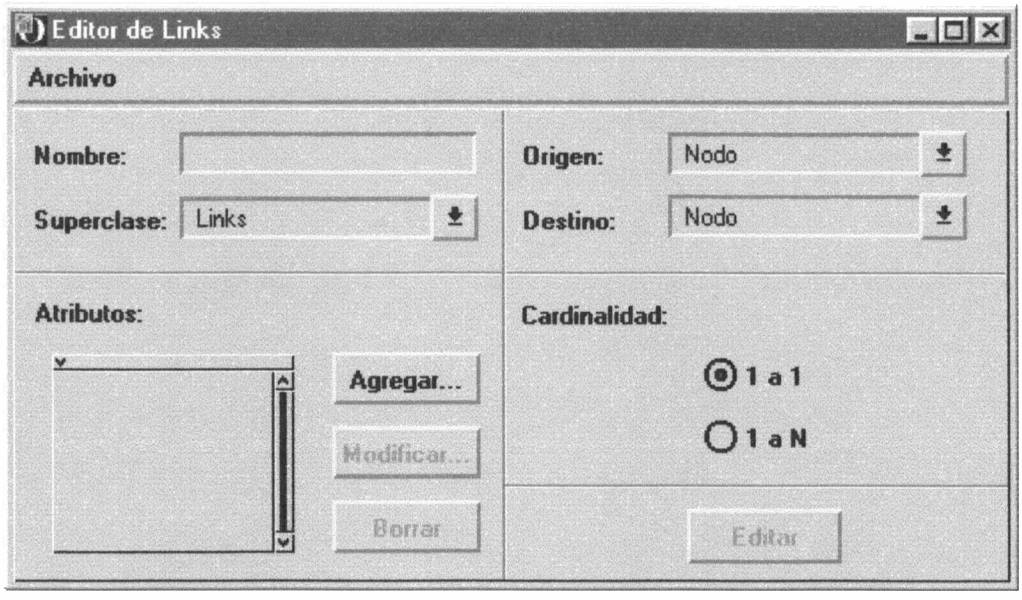


Figura 8. Editor de Links.

El manejo de la lista de atributos también es similar al de editor de nodos, junto a la lista de atributos están los botones *agregar*, *borrar* y *modificar*. Los botones *modificar* y *borrar* estarán deshabilitados si no hay un atributo seleccionado; el botón *borrar* borra el atributo seleccionado sin pedir confirmación; y si existe la perspectiva lo borra de la misma. El botón *agregar* abre un cuadro de diálogo para completar el nombre y el tipo correspondientes al atributo a agregar; en este caso el tipo *anchor* no está disponible porque no tiene sentido que a mitad de un link se tome otro link hacia otro lado. El botón *modificar* abre el mismo cuadro de diálogo, pero con los campos indicando los datos del atributo seleccionado.

Para elegir el *origen* y el *destino* también se dispone de una lista con los nodos navegacionales ya existentes, pero se puede escribir en los campos correspondientes a cada uno un origen o destino no definido aún.

La *cardinalidad* puede ser 1 a 1 o 1 a N. Para el caso de cardinalidad 1 a N no estará disponible la opción *Editar* una perspectiva por más que el tipo de link

tenga atributos; como se explica en la sección 7 (*Posibles Extensiones*) del informe.

El botón *Editar* permite diseñar cómo se verá el nodo intermedio para el caso en que el link tuviese atributos. El editor al que se accede al presionar este botón es el mismo editor de perspectivas que el del editor de nodos, descripto anteriormente en la sección 3.1.

5. Editor de Subsistemas

En el Editor de Subsistemas (figura 9) se declaran los subsistemas del proyecto; diciendo cuál es su nombre, las clases que lo integran y los puntos de entrada. Un punto de entrada es un nodo navegacional al que se puede acceder desde un nodo no perteneciente al subsistema.

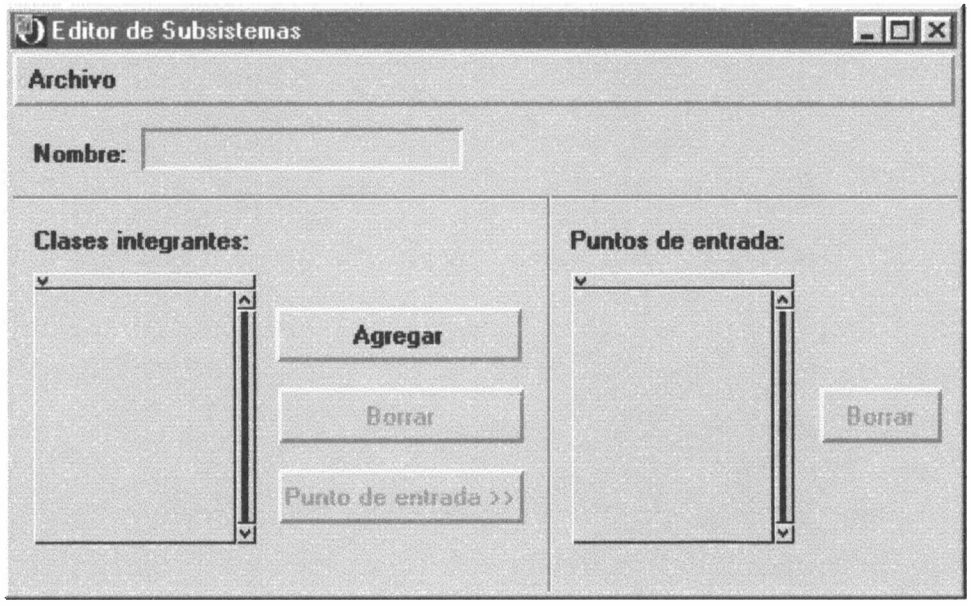


Figura 9. Editor de Subsistemas.

Este editor tiene el menú *Archivo* con las siguientes opciones: *Nuevo*, *Abrir*, *Importar* (permite importar subsistemas de otros proyectos), *Salvar*, *Borrar* y *Salir*.

La lista de clases integrantes tiene a su derecha los botones: *agregar, borrar* y *punto de entrada*. El botón agregar abre un cuadro de diálogo con una lista de los nodos navegacionales existentes que permite seleccionar cuál es el que se desea agregar. El botón *borrar* elimina de la lista la clase seleccionada (automáticamente lo borra también de la lista de puntos de entrada). El botón *punto de entrada* agrega la clase seleccionada a la lista de puntos de entrada.

La lista de puntos de entrada está acompañada por el botón *borrar* que elimina el punto de entrada seleccionado de la misma.

6. Editor de Estructuras de Acceso

Este editor permite definir las estructuras de acceso de la aplicación (figura 10). Debiendo indicarse el *nombre*, el *destino*, el *selector* y si se lo desea un *predicado*.

Este editor tiene el menú *Archivo* con las siguientes opciones: *Nuevo*, *Abrir*, *Importar* (permite importar estructuras de acceso de otros proyectos, importar una estructura de acceso también importa el nodo navegacional destino), *Salvar*, *Borrar* y *Salir*.

El *destino* se selecciona de una lista con todas las nodos navegacionales existentes, para este caso no se puede escribir un *destino* que no exista ya que como se debe seleccionar un *selector* el nodo navegacional debe existir.

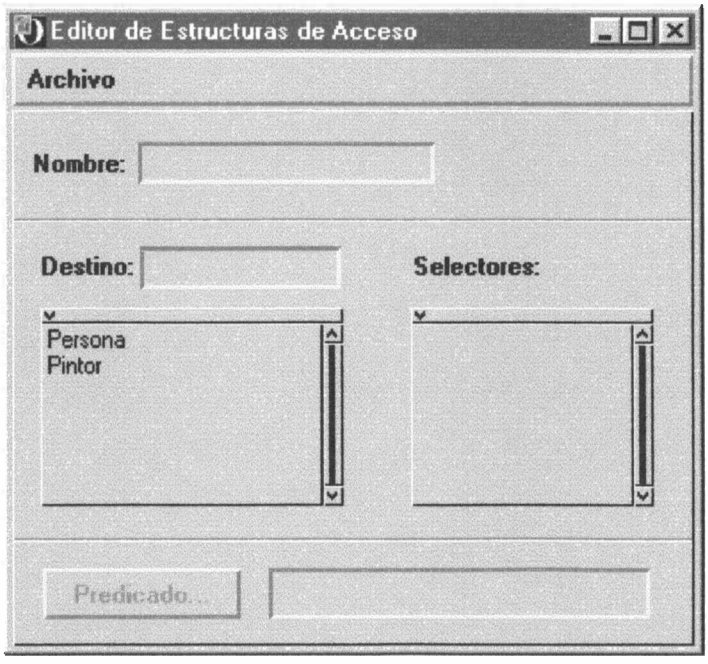


Figura 10. Editor de Estructuras de Acceso.

Una vez seleccionado el *destino* aparece la lista de posibles selectores (atributos del nodo navegacional seleccionado que no son anchors ni gráficos), de esta lista es que se escoge el *selector* deseado.

Para establecer un predicado sobre los destinos se presiona el botón *Predicado* que abre el editor de predicados (sección 6.1). Cuando exista un predicado, éste se verá en el campo a la derecha del botón.

6.1 Editor de Predicado

Las expresiones válidas para un predicado son de la forma: <atributo> <operador> <constante>. Para ello el editor de predicado consta de una lista de *atributos*, un botón *operador* y un campo *constante*. De la lista de atributos se selecciona el atributo deseado, al presionar el botón operador se descuelga un menú con los posibles operadores y la constante simplemente se escribe en el

campo correspondiente. No existe ningún chequeo de que la constante se corresponda con el tipo de atributo seleccionado, esto corre por cuenta del autor.

7. Editor de Hiperbase

El Editor de Hiperbase es el que permite construir la aplicación mediante la instanciación de los nodos navegacionales, links, subsistemas y estructuras de acceso previamente definidos.

La ventana en su parte inferior tiene tres campos; de izquierda a derecha, el primero indica el subsistema en que se está posicionado, cliqueando sobre éste se navega a otro subsistema; el segundo el nodo actual y el total de nodos, si se cliquea aquí se puede navegar a otro nodo del mismo subsistema en que se está posicionado; y el último es una ayuda acerca del objeto seleccionado.

La barra de menú del editor de nodos es muy similar a la del editor de perspectivas (sección 3.1). Las opciones principales son las siguientes: *Archivo*, *Nodo*, *Edición*, *Texto*, *Objeto*, *Dibujo*, *Ventana*. Dentro de *Archivo* están las opciones *nuevo, abrir, salvar, salvar como, borrar* y *salir*; esta última pregunta si se salvan los cambios. *Edición* tiene un submenú con las funciones *cortar*, *copiar* y *pegar* y también *seleccionar Todo*. El submenú de *Texto* tiene las siguientes opciones: *Fuente* abre un cuadro de diálogo para seleccionar el tipo, tamaño y estilo de la fuente, si hay objetos seleccionados la elección se aplica sobre éstos, sino lo que se setea es la fuente por defecto; *Párrafo* que tiene un submenú que permite alinear a izquierda, alinear a derecha, centrar o justificar el texto seleccionado, esta opción no está disponible para el texto de los botones; y las opciones de *negrita, cursiva* y *subrayado*. La opción *Objeto* maneja el nivel de profundidad de los objetos mediante las funciones *traer al frente*, *mandar atrás*, *traer adelante* y *mandar al fondo*. En *Dibujo* se encuentran las opciones *transparente* y *opaco*;

además de *flip horizontal*, *flip vertical*, *rotar izquierda* y *rotar derecha*. También está *alinear* con un submenú con las opciones *arriba, abajo, izquierda, derecha, centrado vertical y centrado horizontal*, esta opción es para aplicar sobre un grupo de objetos, el alineado es respecto al primer objeto seleccionado del grupo. Y por último, *Ventana* que abre la paleta de colores y la paleta de línea, los valores de las paletas se corresponden con el objeto seleccionado y para cambiar el color del background se debe cliquear en éste.

El menú *Nodo* es explicado en este párrafo aparte debido a su vital importancia. La primer opción, *Nuevo Nodo*, es la que permite agregar un nodo a la hiperbase, abre un cuadro de diálogo (figura 11) donde primero se elige el nodo navegacional, luego una perspectiva de éste; si no se desea relacionarlo con una instancia del modelo conceptual, esta información ya es suficiente; caso contrario se deben establecer las relaciones, de la lista de relaciones se selecciona una y en la lista de instancias aparecerán los valores de todas las instancias correspondientes a la relación seleccionada; para el caso de atributos gráficos, la lista es reemplaza por un visor gráfico y dos botones para recorrer los valores; para el caso de atributo anchor lo que aparece en la lista es el valor que identifica al objeto destino y si es 1 a N la lista de valores que lo identifican. Cuando se establece una relación todas las del mismo grupo (es decir, relacionadas con la misma instancia) quedan automáticamente establecidas. Una vez seleccionada la instancia correspondiente se presiona el botón *conectar*; el botón *desconectar todo* elimina **todas** las relaciones establecidas.

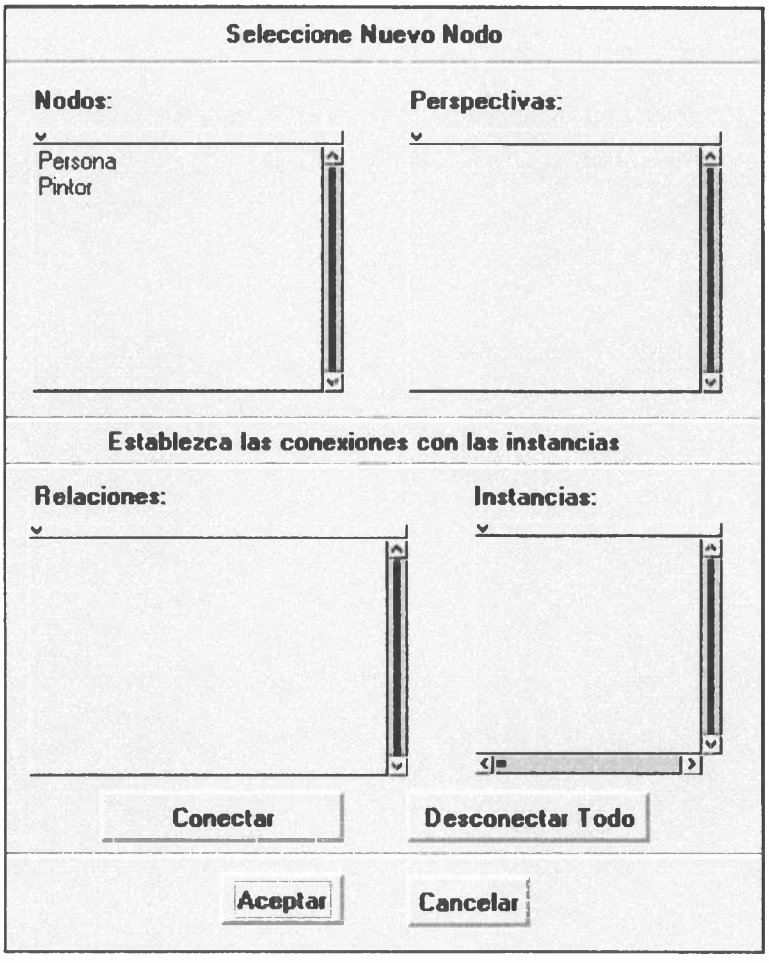


Figura 11. Cuadro de Diálogo para Agregar un Nodo a una Hiperbase.

La opción *nuevo subsistema* abre un diálogo que permite elegir cuál subsistema se va a instanciar. Luego de esto, sólo se podrán instanciar nodos navegacionales que pertenezcan al subsistema instanciado.

Borrar Nodo Actual elimina de la hiperbase el nodo en que se está posicionado. Los links a este nodo desaparecen y los demás nodos son renumerados.

Setear como nodo Inicial establece el nodo actual como el 'primero' de la hiperbase, es decir, el nodo que ve el lector cada vez que se abre la hiperbase.

La opción *Barra de Herramientas*, abre un cuadro de diálogo que sirve para establecer cuáles herramientas de navegación estarán disponibles durante la etapa de navegación, consta de una lista de 'check box' con las opciones de: barra de

herramientas (si está o no la barra); botón atrás; botón inicio; marcas de nodos visitados; next - previous (innecesario si no se ha establecido un tour); y botón de estructuras de acceso globales.

La opción *Definir Tour* permitirá establecer un único tour para la hiperbase, abre un cuadro de diálogo con la lista de nodos a la izquierda y la lista de los pertenecientes al tour a la derecha; cuando se selecciona un nodo éste se verá debajo en el editor de hiperbase (esto permite al autor identificar el nodo); al presionar el botón *agregar* el nodo seleccionado se agregará al final del tour por defecto, si se quiere agregar en algún lugar intermedio de un tour ya creado, se selecciona el nodo del tour que sucederá al que queremos agregar y recién entonces presionamos el botón; el botón *Remover* elimina el nodo seleccionado del tour. Los nodos pueden estar más de una vez en el tour. Se debe tener en cuenta que si se define un tour para la hiperbase se debe habilitar la opción next-previous de la barra de herramientas, por defecto deshabilitada.

La siguiente opción es *Estructuras de acceso*, subdivida en *nueva*, *modificar* y *borrar*. La subopción *nueva*, abre un cuadro de diálogo donde se elige cual estructura de acceso se instanciará, luego de esto se abre un nuevo cuadro de diálogo que permite diseñar su aspecto: color de fondo, color de frente y tipo de fuente; y también, el nombre que llevará. *Modificar* permite rediseñar una estructura de acceso ya creada mediante el mismo cuadro de diálogo y *borrar* elimina una estructura de acceso de la hiperbase. En este caso también debe habilitarse la opción correspondiente en la barra de herramientas.

Ir a Nodo permite navegar a otro nodo ya existente del mismo subsistema en que se está parado; y por último, *Ir a Subsistema*, es la opción que se usa para cambiar a otro subsistema existente.

El Editor de Hiperbase tiene la misma paleta de herramientas que el Editor de Perspectivas, salvo por los botones relacionados con atributos, ya que a esta altura no se pueden agregar nuevos atributos, así como tampoco se pueden ni modifi-

car ni borrar. Sí se pueden agregar, modificar y borrar textos, líneas, rectángulos, etc., tamaño de la ventana, colores y todo el aspecto general de la perspectiva del nodo navegacional. Para ver una descripción de cada elemento de la paleta en particular, dirigirse a la sección 3.1 de este manual.

En el caso que la instanciación del nodo se haya hecho sin relacionarlo con ninguna instancia del modelo conceptual se deberán establecer los valores de los atributos manualmente. Para los valores de los campos se cliquea con el botón derecho sobre el campo seleccionado y aparece el cursor que permite editar el valor deseado. En el caso de atributos Gráficos al cliquear con el botón derecho sobre el seleccionado se abre un "file dialog" (lista de unidades, lista de directorios, lista de archivos, etc.) que permite seleccionar el bitmap deseado. Para el caso de unidades de discos intercambiables, el disco deberá ser colocado antes de invocarlo. Para linkear los atributos Anchor al cliquear con el botón derecho sobre el botón seleccionado se abre un cuadro de diálogo con una lista de todos los posibles destinos según el tipo del link, al seleccionar un elemento de la lista, por debajo se verá cuál es el nodo seleccionado. También se permite cambiar el "label" del botón si así se lo desea; una vez elegido el destino sólo queda apretar el botón *aceptar* y el link estará establecido.

Cuando los nodos de la hiperbase sí estén relacionados con instancias del modelo conceptual, el botón derecho solo servirá para modificar el "label" de los atributos Anchor. Cada vez que se salve la hiperbase, el sistema establecerá automáticamente todos los links posibles; no existe un chequeo de que todos los botones queden linkeados (un botón cuyo nodo destino aún no existe dentro de la hiperbase, obviamente, no podrá linkearse), pero más tarde al recorrer la hiperbase cuando se cliquee en un botón no linkeado no se navegará; para links 1 a N, quedarán establecidos los links para todos los nodos destinos existentes, y funcionarán correctamente durante la etapa de navegación.

8. Navegación

Inicialmente se deberá elegir qué hiperbase es la que se quiere navegar; y para el caso en que ya se estuvo navegando y se guardó un recorrido también se debe elegir el "lector".

Navegar una hiperbase es muy sencillo, sólo con un "click" en los botones nos desplazaremos por los distintos nodos de ella.

Si el autor así la diseñado tendremos una barra de herramientas disponibles en la parte superior, y son las siguientes:

 Abrir: Permite abrir otra hiperbase (o la misma nuevamente) para navegar. Por defecto el recorrido de la hiperbase actual no quedará grabado y tampoco se pide una confirmación al abrir una nueva hiperbase.

 Salir: Cierra la ventana sin pedir ninguna confirmación.

 Grabar: Permite grabar el recorrido, el nodo actual y los nodos visitados de la hiperbase, para poder continuar la navegación en otra ocasión. Aparece un cuadro de diálogo que pide el nombre con que se lo quiere guardar.

 Botón Inicio: Permite navegar al nodo establecido como inicio de la hiperbase.

 Botones previous y next: Estos dos botones son los que permiten recorrer un tour previamente definido por el autor. En principio ambos botones estarán deshabilitados ya que no es posible empezar a recorrer un tour por la mitad. El botón *next* se habilitará cuando se esté parado sobre el nodo que el autor ha

definido como inicio del tour. En cualquier momento del recorrido del tour se podrá salir del mismo navegando a través de los botones del nodo actualmente visible en la ventana, pero luego no se podrá retomar el tour donde habíamos quedado.

 Botón Atrás: Permite navegar al nodo anterior del recorrido.

 Botón Índice: Permite acceder a todas las estructuras de acceso definidas por el autor. Una vez seleccionado el índice aparecerá una ventana con la lista de los nodos a los cuales se puede acceder.

☑ Marca Nodo Visitado: Este es un indicador de si un nodo ha sido visitado o no. Una marca dentro del cuadrado indica que ya se estuvo aquí.

APÉNDICE B. DISEÑO DETALLADO

Las clases diseñadas para el desarrollo de la herramienta son las siguientes:

Principal

Comentario: Aplicación encargada de controlar el menú principal. Maneja los distintos proyectos del sistema, permitiendo crear uno nuevo y abrir o borrar uno ya existe. También se encarga de los editores a través de los botones correspondientes a cada uno.

Superclase: Application Model.

Subclases: ninguna.

Variables de Instancia: proyectos (colección de todos los proyectos del sistema), proye (proyecto activo), editorNodos, editorLinks, editorAccesos, editorSubsistemas, editorHiperbase, editorGrafos y navegacion.

Métodos Principales:

abrirProyecto : abre un cuadro de diálogo que permite elegir cuál es el proyecto que se quiere abrir.

abrir : abre el proyecto elegido en el cuadro de diálogo.

nuevoProyecto : abre un cuadro de diálogo para elegir las características (nombre y categoría de las clases del modelo conceptual) del nuevo proyecto.

nuevo:con: : crea el nuevo proyecto con el nombre y la categoría elegidos y lo pone como proyecto activo.

borrarProyecto : abre un cuadro de diálogo para seleccionar cuál proyecto se quiere eliminar del sistema.

borrar : elimina del sistema el proyecto seleccionado.

editNodos : abre el editor de nodos.

editLinks : abre el editor de links.

editAcceso : abre el editor de estructuras de acceso.

editsubsistemas : abre el editor de subsistemas.

editgrafos : abre el editor de esquemas.

autoría : abre el editor de hiperbase.

lector : abre un cuadro de diálogo que permite elegir cuál hiperbase se quiere navegar.

salir : sale del sistema.

initialize : inicializa todas las variables del sistema.

EditorNodos

Comentario: Interface para el manejo de los nodos navegacionales, permitiendo crear uno nuevo (definiendo su nombre, su superclase, sus atributos, las distintas perspectivas y su relación con el modelo conceptual) y abrir o borrar uno ya existente.

Superclase: Application Model.

Subclases: ninguna.

Variables de Instancia: nodos (colección de todos los nodos del proyecto), nombre (nombre del nodo), superclase (superclase del nodo actual), atributos (colección de los atributos del nodo con todas sus propiedades), relaciones (conjunto de relaciones atributo-clase-variable de instancia), perspectivas (colección de las perspectivas del nodo), lissuper (lista de los nombres de los nodos del proyecto actual de la cual se puede elegir la superclase del nodo), lisAtrib (lista de los nombres de los atributos del nodo), lisPer (lista de las perspectivas del nodo), y editorPerps (editor de perspectivas).

Métodos Principales:

nuevo : crea un nuevo nodo navegacional. Limpia los fields y listas del editor.

abrir : abre un cuadro de diálogo para seleccionar el nodo a abrir.

abrir: : rellena los fields y listas del editor con los datos del nodo elegido para abrir.

borrarNodo : abre un cuadro de diálogo que permite seleccionar cuál nodo se desea eliminar del proyecto.

borrar: : elimina el nodo seleccionado en el cuadro de diálogo del proyecto.

salvar : salva los datos del nodo actual, verificando que la definición del nodo sea correcta.

importar : abre un cuadro de diálogo que permite elegir desde qué proyecto y cuáles nodos se desean importar.

importar:con: : agrega los nodos seleccionados al proyecto actual y actualiza el editor con los datos del primero de la lista.

importar2:con: : agrega los nodos recibidos al proyecto actual sin actualizar el editor, usado cuando lo que se importa en realidad es una estructura de acceso.

abrirGraf: : abre el editor de nodos con los datos provenientes del editor de esquemas.

verificar : usado para verificar y preguntar si hay cambios sin salvar ante un cambio de estado del editor.

salir : cierra el editor de nodos.

agregar : abre un cuadro de diálogo que permite ingresar las características del atributo que se desea agregar al nodo.

agregar: : agrega el atributo recibido al nodo actual.

borrar : elimina del nodo actual el atributo seleccionado en la lista de nombres de atributos.

modificar : permite modificar los datos del atributo seleccionado en la lista de nombres de atributos.

agregarPer : abre el editor de perspectivas para diseñar una nueva perspectiva del nodo.

agregarPer: : agrega la perspectiva recibida al nodo actual.

borrarPer : elimina la perspectiva seleccionada en la lista del nodo actual.

modificarPer : permite modificar la perspectiva seleccionada en la lista abriendo el editor de perspectivas con el diseño de la misma.

relacionar : abre una ventana que permite establecer las relaciones entre este nodo y el modelo conceptual.

relacionar: : setea las relaciones del nodo actual al conjunto de relaciones recibidas.

lista : muestra la lista de posibles superclases del nodo actual.

EditorLinks

Comentario: Interface para el manejo de los links del proyecto, permitiendo crear uno nuevo (definiendo su nombre, su superclase, su origen, su destino, su cardinalidad, sus atributos y perspectiva si corresponde) y abrir o borrar uno ya existente.

Superclase: Application Model.

Subclases: ninguna.

Variables de Instancia: links (colección de todos los links del proyecto), nombre (nombre del link), superclase (superclase del link actual), atributos (colección de los atributos del link con todas sus propiedades), origen (nodo origen del tipo de link actual), destino (nodo destino del tipo de link actual), cardi (cardinalidad del link), perspectiva (perspectiva del link), lisorigen (lista de nodos origen posibles), lisdestino (lista de nodos destino posibles), lissuper (lista de los nombres de los links del proyecto actual de la cual se puede elegir la superclase del link), lisAtrib (lista de los nombres de los atributos del link) y editorPerps (editor de perspectivas).

Métodos Principales:

nuevo : crea un nuevo tipo de link. Limpia los fields y listas del editor.

abrir : abre un cuadro de diálogo para seleccionar el link que se desea abrir.

abrir: : rellena los fields y listas del editor con los datos del link elegido para abrir.

borrarLink : abre un cuadro de diálogo que permite seleccionar cuál link se desea eliminar del proyecto.

borrar: : elimina el link seleccionado en el cuadro de diálogo del proyecto.

salvar : salva los datos del link actual, verificando que la definición del link sea correcta.

importar : abre un cuadro de diálogo que permite elegir desde qué proyecto y cuáles tipos de link se desean importar.

importar:con: : agrega los links seleccionados al proyecto actual y actualiza el editor con los datos del primero de la lista.

abrirLink: : abre el editor de links con los datos provenientes del editor de esquemas.

verificar : usado para verificar y preguntar si hay cambios sin salvar ante un cambio de estado del editor.

salir : cierra el editor de links.

agregar : abre un cuadro de diálogo que permite ingresar las características del atributo que se desea agregar al link.

agregar: : agrega el atributo recibido al link actual.

borrar : elimina del link actual el atributo seleccionado en la lista de nombres de atributos.

modificar : permite modificar los datos del atributo seleccionado en la lista de nombres de atributos.

editar : abre el editor de perspectivas para diseñar la perspectiva del link.

agregarPer: : agrega la perspectiva recibida al link actual.

lista : muestra la lista de posibles superclases del link actual.

listaOr : muestra la lista de nodos origen posibles.

listaDe : muestra la lista de nodos destino posibles.

EditorAcceso

Comentario: Interface para el manejo de las estructuras de acceso del proyecto, permitiendo crear uno nuevo (definiendo su nombre, su nodo destino, su atributo selector y su predicado) y abrir o borrar uno ya existente.

Superclase: Application Model.

Subclases: ninguna.

Variables de Instancia: accesos (colección de todos las estructuras de acceso del proyecto), nombre (nombre de la estructura de acceso), destino (nodo destino de la estructura de acceso actual), predicado (predicado de la estructura de acceso), lisSelect (lista de los posibles atributos selectores) y lisdestino (lista de los posibles nodos destino).

Métodos Principales:

nuevo : crea una nueva estructura de acceso. Limpia los fields y listas del editor.

abrir : abre un cuadro de diálogo para seleccionar la estructura de acceso que se desea abrir.

abrir: : rellena los fields y actualiza las listas del editor con los datos de la estructura de acceso elegida para abrir.

borrarAcceso : abre un cuadro de diálogo que permite seleccionar cuál estructura de acceso se desea eliminar del proyecto.

borrar: : elimina la estructura de acceso seleccionada en el cuadro de diálogo del proyecto.

salvar : salva los datos de la estructura de acceso actual, verificando que la definición de la misma sea correcta.

importar : abre un cuadro de diálogo que permite elegir desde qué proyecto y cuáles estructuras de acceso se desean importar.

importar:con: : agrega las estructuras de acceso seleccionadas al proyecto actual y actualiza el editor con los datos de la primera de la lista.

abrirAcceso: : abre el editor de estructuras de acceso con los datos provenientes del editor de esquemas.

verificar : usado para verificar y preguntar si hay cambios sin salvar ante un cambio de estado del editor.

salir : cierra el editor de estructuras de acceso.

predicar : abre el editor de predicados.

predicado : setea el predicado recibido a la estructura de acceso global.

EditorSubsistemas

Comentario: Interface para el manejo de los subsistemas del proyecto, permitiendo crear uno nuevo (definiendo su nombre, sus clases integrantes y sus puntos de entrada) y abrir o borrar uno ya existente.

Superclase: Application Model.

Subclases: ninguna.

Variables de Instancia: subsistemas (colección de todos los subsistemas del proyecto), nombre (nombre del subsistema), lisClases (lista de las clases que integran el subsistema) y lisEntry (lista de los puntos de entrada del subsistema).

Métodos Principales:

nuevo : crea un nuevo subsistema. Limpia los fields y listas del editor.

abrir : abre un cuadro de diálogo para seleccionar el subsistema a abrir.

abrir: : rellena los fields y listas del editor con los datos del subsistema elegido para abrir.

borrarSubsi : abre un cuadro de diálogo que permite seleccionar cuál subsistema se desea eliminar del proyecto.

borrar: : elimina el subsistema seleccionado en el cuadro de diálogo del proyecto.

salvar : salva los datos del subsistema actual, verificando que la definición del subsistema sea correcta.

importar : abre un cuadro de diálogo que permite elegir desde qué proyecto y cuáles subsistemas se desean importar.

importar:con: : agrega los subsistemas seleccionados al proyecto actual y actualiza el editor con los datos del primero de la lista.

verificar : usado para verificar y preguntar si hay cambios sin salvar ante un cambio de estado del editor.

salir : cierra el editor de subsistemas.

agregar : abre un cuadro de diálogo que permite seleccionar que nodo navegacional se desea agregar al subsistema.

agregar: : agrega el nodo navegacional seleccionado al subsistemas.

borrar : elimina el nodo navegacional seleccionado en la lista de clases del subsistema.

agEntry : agrega el nodo navegacional seleccionado en la lista de clases a la lista de puntos de entrada.

borrar2 : elimina el punto de entrada seleccionado de la lista correspondiente del subsistema.

EditorGrafos

Comentario: Interface para diagramar un esquema de la aplicación a realizar.

Superclase: Application Model.

Subclases: ninguna.

Variables de Instancia: grafos (colección de todos los esquemas del proyecto), grafoRec (esquema actual), toolPalette (paleta de herramientas del editor de esquemas) y selection (colección de objetos seleccionados).

Métodos Principales:

nuevo : crea un nuevo esquema. Limpia la pantalla del editor de esquemas.

abrir : abre un cuadro de diálogo para seleccionar el esquema que se desea abrir.

abrir: : rellena la pantalla del editor con los componentes del esquema elegido para abrir.

borrar : abre un cuadro de diálogo que permite seleccionar cuál esquema se desea eliminar del proyecto.

borrar: : elimina el esquema seleccionado en el cuadro de diálogo del proyecto.

guardar : salva los datos del esquema actual, pidiendo el nombre si es necesario.

guardarComo : abre un cuadro de diálogo que permite grabar el esquema con otro nombre.

guardarComo: : guarda el esquema con el nombre recibido.

verificar : usado para verificar y preguntar si hay cambios sin salvar ante un cambio de estado del editor.

salir : cierra el editor de esquemas.

EditorHiperbase

Comentario: Interface para crear una hiperbase, a través de la instanciación de los nodos, links y estructuras de acceso previamente definidos.

Superclase: EditorPerspectivas.

Subclases: ninguna.

Variables de Instancia: hiperbases (colección de todas las hiperbases del proyecto), hiperActiva (hiperbase actualmente activa) y subActual (subsistema activo).

Métodos Principales:

nuevo : crea una nueva hiperbase, limpiando la pantalla del editor de hiperbases.

abrir : abre un cuadro de diálogo para seleccionar la hiperbase que se desea abrir.

abrir: : actualiza las variables y la pantalla del editor con los datos de la hiperbase elegida para abrir.

borrar : abre un cuadro de diálogo que permite seleccionar cuál hiperbase se desea eliminar del proyecto.

borrar: : elimina la hiperbase seleccionada en el cuadro de diálogo del proyecto.

salvar : salva los datos de la hiperbase actual, pidiendo el nombre si es necesario.

salvarComo : abre un cuadro de diálogo que permite salvar la hiperbase con otro nombre.

salvarComo: : guarda la hiperbase con el nombre recibido.

verificar : usado para verificar y preguntar si hay cambios sin salvar ante un cambio de estado del editor.

salir : cierra el editor de hiperbases.

nuevoNodo : abre un cuadro de diálogo que permite seleccionar el nodo navegacional y la correspondiente perspectiva, además de la instancia del modelo conceptual con la que se lo quiere relacionar (si así lo desea).

nuevoNodo:con: : crea una instancia del nodo navegacional recibido usando la perspectiva recibida. Actualizando la ventana del editor de hiperbases con estos datos.

nuevoSub : abre un cuadro de diálogo que permite seleccionar el subsistema que se desea instanciar.

nuevoSub: : crea el nuevo subsistema, actualizando subActual y la ventana del editor.

borrarNodo : luego de pedir la confirmación elimina de la hiperbase el nodo actual en que se está posicionado.

borrarNodo: : elimina de la hiperbase el nodo cuyo número es recibido (interno).

Navegacion

Comentario: Interface para navegar una hiperbase.

Superclase: ApplicationModel.

Subclases: ninguna.

Variables de Instancia: hiperActiva (hiperbase que se está navegando), perRec (nodo actual de la hiperbase), historia (recorrido a través de los nodos), visitados (colección de nodos visitados), version (versión de la hiperbase que se está recorriendo, lector) y lisInd (lista de estructuras de acceso globales).

Métodos Principales:

abrir : abre un cuadro de diálogo para seleccionar la hiperbase que se desea navegar.

abrir: : actualiza la pantalla del editor con los datos de la hiperbase elegida para navegar.

abrir:en: : actualiza la pantalla del editor con los datos de la hiperbase y versión elegida para navegar.

inicio : navega al nodo inicial de la hiperbase activa.

atras : retrocede un nodo en la historia de la hiperbase.

next : navega al próximo nodo del tour de la hiperbase.

previous : navega al nodo anterior en el tour de la hiperbase.

salvarVer : salva el estado actual del recorrido de la hiperbase, pidiendo un nombre de versión si es necesario.

salvarVer: : salva el estado actual del recorrido de la hiperbase bajo el nombre de versión recibido.

irA: : navega al nodo recibido (interno).

botonPresionao: : recibe el punto donde se presionó el botón del mouse, verifica si hay algún botón allí, y si es así, navega al destino del mismo.

salir : cierra el ambiente de navegación.

EditorPerspectivas

Comentario: Interface para diseñar las perspectivas de los nodos navegacionales y la perspectiva de los links con atributos.

Superclase: ApplicationModel.

Subclases: EditorHiperbase.

Variables de Instancia: perRec (perspectiva actual que se está diseñando), fuente (fuente general) y toolPalette (paleta de herramientas del editor de perspectivas).

Métodos Principales:

salvar : salva el estado actual de la perspectiva.

salir : cierra el editor de perspectivas, si hay cambios pide confirmación.

abrir:on: : abre una nueva perspectiva con el nombre recibido.

abrirDirect:on: : abre una perspectiva ya existente.

abrir:with:on: : abre una nueva perspectiva con el nombre recibido y con los componentes de la perspectiva.

fuentes : abre un cuadro de diálogo que permite seleccionar fuente, tamaño y estilo de un texto.

colorPalette : abre la paleta de colores.

linePalette : abre la paleta de ancho de línea.

ViewGeneral

Comentario: Clase abstracta que reúne el comportamiento común de las distintas 'views' de los editores.

Superclase: CompositeView.

Subclases: EditorView, LectorView, HiperView.

Variables de Instancia: portapapeles (colección de los componentes copiados o borrados), selection (colección de los componentes seleccionados) y selectPoint (punto de la ventana donde se presionó el botón del mouse).

Métodos Principales:

copiar : copia los elementos seleccionados al portapapeles.

pegar : copia los elementos del portapapeles en la ventana.

cortar : copia los elementos seleccionados al portapapeles y los elimina de la ventana.

selectAll : selecciona todos los componentes de la ventana.

juziza : justifica el texto de los componentes seleccionados a izquierda.

juzder : justifica el texto de los componentes seleccionados a derecha.

juzcen : centra el texto de los componentes seleccionados.

juztot : justifica totalmente el texto de los componentes seleccionados.

negrita : setea el texto de los componentes seleccionados a negrita.

cursiva : setea el texto de los componentes seleccionados a cursiva.

subraya : setea el texto de los componentes seleccionados a subrayado.

toVisualFrontOneNotch : trae los componentes seleccionados un nivel hacia al frente.

toVisualBackOneNotch : envía los componentes seleccionados un nivel hacia al fondo.

toVisualFront : trae los componentes seleccionados al frente.

toVisualBack : envía los componentes seleccionados al fondo.

transparente : hace transparentes los componentes seleccionados.

opaco : hace opacos los componentes seleccionados.

flipV : convierte los componentes seleccionados a su simétrico con respecto al eje vertical.

flipH : convierte los componentes seleccionados a su simétrico con respecto al eje horizontal.

rotarI : rota 90° los componentes seleccionados a izquierda.

rotarD : rota 90° los componentes seleccionados a derecha.

alinearAbaj : alinea los componentes seleccionados al borde inferior del primer componente seleccionado.

alinearArr : alinea los componentes seleccionados al borde superior del primer componente seleccionado.

alinearDer : alinea los componentes seleccionados al borde derecho del primer componente seleccionado.

alinearIzq : alinea los componentes seleccionados al borde izquierdo del primer componente seleccionado.

centradoHor : centra los componentes seleccionados con respecto al eje horizontal.

centradoVer : centra los componentes seleccionados con respecto al eje vertical.

movAbj : mueve los elementos seleccionados una unidad hacia abajo.

movArr : mueve los elementos seleccionados una unidad hacia arriba.

movDer : mueve los elementos seleccionados una unidad hacia la derecha.

movIzq : mueve los elementos seleccionados una unidad hacia la izquierda.

cambioColor : setea el color de los elementos seleccionados al nuevo color de background o foreground según corresponda.

cambioLínea : setea el ancho de línea de los componentes seleccionados al nuevo valor.

lineaInit: : prepara el editor para dibujar una línea con origen en el punto recibido como parámetro.

linea: : dibuja la línea usando el punto recibido como fin de la misma.

lineaFin : termina el dibujo de la línea.

rectanguloInit: : prepara el editor para dibujar un rectángulo con origen en el punto recibido como parámetro.

rectangulo: : dibuja el rectángulo usando el punto recibido como esquina opuesta del mismo.

rectanguloFin : termina el dibujo del rectángulo.

elipseInit: : prepara el editor para dibujar una elipse con origen en el punto recibido como parámetro.

elipse: : dibuja la elipse usando el punto recibido como esquina opuesta del rectángulo que contiene a la misma.

elipseFin : termina el dibujo de la elipse.

textoInit: : prepara el editor para dibujar un rectángulo que contendrá texto con origen en el punto recibido como parámetro.

texto: : dibuja el rectángulo que contendrá el texto usando el punto recibido como esquina opuesta del mismo.

textoFin : termina el dibujo del rectángulo del texto.

imagenInit: : prepara el editor para dibujar un rectángulo que contendrá una imagen con origen en el punto recibido como parámetro.

imagen: : dibuja el rectángulo que contendrá la imagen usando el punto recibido como esquina opuesta del mismo.

imagenFin : termina el dibujo del rectángulo de la imagen y abre un cuadro de diálogo que permite elegir cuál es la imagen.

curvaInit: : prepara el editor para dibujar una curva con origen en el punto recibido como parámetro.

curva: : dibuja la curva usando el punto recibido como fin de la misma.

curvaFin : termina el dibujo de la curva.

lineaConInit: : prepara el editor para dibujar líneas consecutivas con origen de la primer línea en el punto recibido como parámetro.

líneaConInter: : dibuja la línea usando el punto recibido como fin de la misma y comienzo de la próxima.

líneaCon: : dibuja la última línea usando el punto recibido como fin de la misma.

líneaConFin : termina el dibujo de líneas consecutivas.

poligonoInit: : prepara el editor para dibujar un polígono irregular con primer vértice en el punto recibido como parámetro.

líneaConInter: : agrega un nuevo vértice al polígono en el punto recibido.

líneaCon: : agrega el último vértice del polígono.

líneaConFin : termina el dibujo del polígono.

imagenAtribInit: : prepara el editor para dibujar un rectángulo que contendrá un atributo de tipo gráfico con origen en el punto recibido.

imagenAtrib: : dibuja el rectángulo usando como esquina opuesta el punto recibido.

imagenAtribFin : termina el dibujo del rectángulo y abre un cuadro de diálogo que permite seleccionar cuál es el atributo gráfico correspondiente.

textoAtribInit: : prepara el editor para dibujar un rectángulo que contendrá un atributo de tipo string, número o fecha con origen en el punto recibido.

textoAtrib: : dibuja el rectángulo usando como esquina opuesta el punto recibido.

textoAtribFin : termina el dibujo del rectángulo y abre un cuadro de diálogo que permite seleccionar cuál es el atributo string, número o fecha correspondiente.

botonInit: : prepara el editor para dibujar un botón con origen en el punto recibido.

boton: : dibuja el botón usando como esquina opuesta el punto recibido.

botonFin : termina el dibujo del botón y abre un cuadro de diálogo que permite seleccionar cuál es el atributo anchor correspondiente.

selectInit: : determina si corresponde mover un objeto, ajustar el tamaño de un objeto o dibujar un cuadrado que permite seleccionar varios objetos a la vez, según el punto recibido corresponde a un objeto, un rectangulito o a nada.

select: : según lo determinado anteriormente decide qué es lo que hay que hacer.

move: : mueve la selección al punto recibido.

resize: : ajusta el tamaño del objeto seleccionado según el punto recibido.

selectFin : termina el move o resize dejando seleccionado los objetos que se han movido o el objeto al que se le ha ajustado el tamaño.

rectSelectFin : termina de dibujar el rectángulo para elegir cuáles objetos se seleccionan, selecciona todos los que estén dentro del mismo y borra el rectángulo.

EditorView

Comentario: clase *view* correspondiente al Editor de Perspectivas.

Superclase: ViewGeneral.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

defaultControllerClass : devuelve cuál es la clase *Controller* apropiada para esta *view*.

LectorView

Comentario: clase *view* correspondiente a la herramienta de navegación.

Superclase: ViewGeneral.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

defaultControllerClass : devuelve cuál es la clase *Controller* apropiada para esta *view*.

HiperView

Comentario: clase *view* correspondiente al editor de hiperbases.

Superclase: ViewGeneral.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

defaultControllerClass : devuelve cuál es la clase *Controller* apropiada para esta *view*.

copiar, *cortar* : redefine estos métodos ya que no se pueden borrar objetos relacionados con atributos.

GrafoView

Comentario: clase *view* correspondiente al Editor de Esquemas.

Superclase: CompositeView.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

defaultControllerClass : devuelve cuál es la clase *Controller* apropiada para esta *view*.

nodoInit: : prepara el editor para dibujar un nodo con origen en el punto recibido como parámetro.

nodo: : dibuja el nodo usando el punto recibido como origen del mismo.

nodoFin : termina el dibujo del nodo y abre un cuadro de diálogo que permite ingresar el nombre del nodo.

linkInit: : prepara el editor para dibujar un link con origen en el nodo correspondiente al punto recibido como parámetro.

linkInter: : dibuja el link hasta el nodo correspondiente al punto recibido como parámetro y abre un cuadro de diálogo que permite ingresar el nombre del link.

esUnInit: : prepara el editor para dibujar una relación *es-un* con padre en el nodo correspondiente al punto recibido como parámetro.

esUnInter: : dibuja la relación *es-un* hasta el nodo correspondiente al punto recibido como parámetro.

accesoInit: : prepara el editor para dibujar una estructura de acceso con origen en el punto recibido como parámetro.

acceso: : dibuja la estructura de acceso usando el punto recibido como origen del mismo.

accesoFin: : termina el dibujo de la estructura de acceso, la conecta al nodo correspondiente al punto recibido como parámetro y abre un cuadro de diálogo que permite ingresar el nombre de la estructura de acceso.

selectInit: prepara el editor para mover el objeto correspondiente al punto recibido (sólo nodos y estructuras de acceso).

select: mueve el objeto seleccionado al punto recibido.

selectFin : termina de mover el objeto y lo deja seleccionado.

subsistema : crea un nuevo subsistema con los nodos seleccionados y abre un cuadro de diálogo que permite ingresar el nombre del mismo.

editarFin: : abre el editor correspondiente al objeto al cuál el punto recibido pertenece.

ControllerGeneral

Comentario: Clase abstracta que reúne el comportamiento común de las distintos 'controllers' de los editores.

Superclase: ModalController.

Subclases: EditorController, LectorController, HiperController, GrafoController.

Variables de Instancia: *first, continue* (variables de control).

Métodos Principales:

controlInitialize : redefine el método para setear las variables *first* y *continue*.

isControlActive : redefine el método para determinar cuando el controller está activo.

controlActivity : determina cuál es la actividad que hay que controlar (botón izquierdo, botón derecho, el teclado, etc.).

redButtonActivity : responsabilidad de la subclases, que se debe hacer cuando se presiona el botón izquierdo del mouse.

keyboardActivity : controla la actividad del teclado (responde a la tecla DELETE, y a las flechas), según la tecla le avisa a la correspondiente view lo acontecido.

multiActivity : responsabilidad de la subclases, que se debe hacer cuando se presiona el botón izquierdo del mouse y *continue* es true.

dobleClickControl : responsabilidad de la subclases, que se debe hacer cuando se presiona el botón derecho del mouse.

releaseActivity : responsabilidad de la subclases, que se debe hacer cuando se suelta el botón izquierdo del mouse.

EditorController

Comentario: Clase *controller* correspondiente al editor de perspectivas.

Superclase: ControllerGeneral.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

redButtonActivity : controla si el botón izquierdo del mouse está presionado, y según qué acción esté seleccionada en la paleta del editor de perspectivas le comunica a la *view* qué se debe hacer.

multiActivity : controla cada vez que se volvió a presionar el botón izquierdo sin presionar el derecho a través de la variable *continue* usado para dibujar polígonos y líneas consecutivas.

dobleClickControl : controla el botón derecho del mouse, le avisa a la *view* que se presionó este botón salvo que se esté dibujando un polígono o líneas consecutivas, en este caso le comunica que el dibujo llegó a su fin.

releaseActivity : controla si el botón izquierdo del mouse fue soltado, también para dibujar polígonos y líneas consecutivas, le comunican a la *view* que agregue un nuevo vértice.

controlTerminate : avisa a la *view* que se terminó con lo que se estaba haciendo.

LectorController

Comentario: Clase *controller* correspondiente a la herramienta de navegación.

Superclase: ControllerGeneral.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

isControlActive : redefine el método de cuando hay actividad para controlar (sólo el botón izquierdo del mouse).

controlActivity : controla si el botón izquierdo del mouse está apretado o no.

redButtonActivity : avisa a la *view* que se presionó el botón del mouse.

noredButtonActivity : avisa a la *view* que se soltó el botón del mouse.

HiperController

Comentario: Clase *controller* correspondiente al editor de hiperbases.

Superclase: ControllerGeneral.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

redButtonActivity : controla si el botón izquierdo del mouse está presionado, y según qué acción esté seleccionada en la paleta del editor de hiperbases le comunica a la *view* qué se debe hacer.

multiActivity : controla cada vez que se volvió a presionar el botón izquierdo sin presionar el derecho a través de la variable *continue* usado para dibujar polígonos y líneas consecutivas.

dobleClickControl : controla el botón derecho del mouse, le avisa a la *view* que se presionó este botón salvo que se esté dibujando un polígono o líneas consecutivas, en este caso le comunica que el dibujo llegó a su fin.

releaseActivity : controla si el botón izquierdo del mouse fue soltado, también para dibujar polígonos y líneas consecutivas, le comunican a la *view* que agregue un nuevo vértice.

controlTerminate : avisa a la *view* que se terminó con lo que se estaba haciendo.

GrafoController

Comentario: Clase *controller* correspondiente al editor de esquemas.

Superclase: ControllerGeneral.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

redButtonActivity : controla si el botón izquierdo del mouse está presionado, y según qué acción esté seleccionada en la paleta del editor de esquemas le comunica a la *view* qué se debe hacer.

multiActivity : controla cada vez que se volvió a presionar el botón izquierdo sin presionar el derecho a través de la variable *continue* usado para dibujar las estructuras de acceso.

releaseActivity : controla si el botón izquierdo del mouse fue soltado, también para estructuras de acceso.

controlTerminate : avisa a la *view* que se terminó con lo que se estaba haciendo.

ToolPalette

Comentario: paleta de herramientas correspondiente al editor de perspectivas.

Superclase: ApplicationModel.

Subclases: HiperPalette.

Variables de Instancia: *accion* (indica cuál es la acción activa) y *modelo* (indica quién recibe los mensajes).

Métodos Principales:

abrir: : abre la paleta de herramientas para el modelo recibido.

select : setea seleccionar como acción activa, actualiza el cursor y avisa al modelo del cambio.

linea : setea dibujar línea como acción activa, actualiza el cursor y avisa al modelo del cambio.

cuadrado : setea dibujar rectángulo como acción activa, actualiza el cursor y avisa al modelo del cambio.

circulo : setea dibujar elipse como acción activa, actualiza el cursor y avisa al modelo del cambio.

curva : setea dibujar curva como acción activa, actualiza el cursor y avisa al modelo del cambio.

lineaCon : setea dibujar líneas consecutivas como acción activa, actualiza el cursor y avisa al modelo del cambio.

irregular : setea dibujar polígono como acción activa, actualiza el cursor y avisa al modelo del cambio.

text : setea dibujar rectángulo de texto como acción activa, actualiza el cursor y avisa al modelo del cambio.

imagen : setea dibujar imagen como acción activa, actualiza el cursor y avisa al modelo del cambio.

imagenAtrib : setea dibujar atributo gráfico como acción activa, actualiza el cursor y avisa al modelo del cambio.

field : setea dibujar rectángulo de texto para atributos como acción activa, actualiza el cursor y avisa al modelo del cambio.

push : setea dibujar botón como acción activa, actualiza el cursor y avisa al modelo del cambio.

buttonDown: : reúne las acciones comunes de los botones anteriormente mencionados.

HiperPalette

Comentario: paleta de herramientas correspondiente al editor de hiperbase.

Superclase: ToolPalette.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

buttonDown: : reúne las acciones comunes de los botones anteriormente mencionados.

disableTodo: : deshabilita todos los botones de la paleta de herramientas.

enableTodo: : habilita todos los botones de la paleta de herramientas.

GrafoPalette

Comentario: paleta de herramientas correspondiente al editor de esquemas.

Superclase: ApplicationModel.

Subclases: ninguna.

Variables de Instancia: accion (indica cuál es la acción activa de la paleta), mostrarH (valor booleano que indica si debe mostrarse la ayuda de cómo dibujar un link) y mostrarA (valor booleano que indica si debe mostrarse la ayuda de cómo dibujar una estructura de acceso).

Métodos Principales:

select : setea seleccionar como acción activa, actualiza el cursor y avisa al modelo del cambio.

nodo : setea dibujar nodo como acción activa, actualiza el cursor y avisa al modelo del cambio.

link : setea dibujar link como acción activa, actualiza el cursor y avisa al modelo del cambio.

acceso : setea dibujar estructura de acceso como acción activa, actualiza el cursor y avisa al modelo del cambio.

esun : setea dibujar relación es-un como acción activa, actualiza el cursor y avisa al modelo del cambio.

editar : setea editar objeto como acción activa, actualiza el cursor y avisa al modelo del cambio.

subsistema : avisa al modelo que dibuje un subsistema con los nodos seleccionados.

buttonDown: : reúne las acciones comunes de los botones anteriormente mencionados.

Componente

Comentario: clase abstracta que reúne el comportamiento común de todas las figuras gráficas de los editores.

Superclase: SimpleComponent.

Subclases: Curva, Elipse, Imagen, Linea, MultiLinea y Rectangulo.

Variables de Instancia: *origen* (punto origen de la figura), *corner* (punto extremo de la figura) y *ancho* (ancho de las líneas de la figura).

Métodos Principales:

initialize : inicializa la figura con los colores y ancho de las paletas correspondientes.

opaco : transforma el componente a opaco.

transparente : transforma el componente a transparente.

containsPoint: :testea si el componente contiene al punto recibido.

origen : retorna el origen del componente.

origen: : setea el origen del componente al punto recibido.

corner : retorna el corner del componente.

corner: : setea el corner del componente al punto recibido.

ancho : retorna el ancho de línea del componente.

ancho: : setea el ancho de línea del componente al valor recibido.

foreColor : retorna el color de frente del componente.

foreColor: : setea el color de frente del componente al color recibido.

backColor : retorna el color de fondo del componente.

backColor: : setea el color de fondo del componente al color recibido.

Linea

Comentario: clase que representa una línea.

Superclase: Componente.

Subclases: LinkGrafico.

Variables de Instancia: ninguna.

Métodos Principales:

containsPoint: :redefine este método que testea si la línea contiene al punto recibido.

displayOn: : dibuja la línea en el *graphicsContext* recibido.

flipH : transforma la línea según el eje horizontal.

flipV : transforma la línea según el eje vertical.

rotarI : rota la línea 90 grados a izquierda.

rotarD : rota la línea 90 grados a derecha.

Curva

Comentario: clase que representa una curva.

Superclase: Componente.

Subclases: ninguna.

Variables de Instancia: *stang* (ángulo de comienza de la línea) y *swang* (ángulo de barrido de la curva).

Métodos Principales:

containsPoint: :redefine este método que testea si la curva contiene al punto recibido.

displayOn: : dibuja la curva en el *graphicsContext* recibido.

flipH : transforma la curva según el eje horizontal.

flipV : transforma la curva según el eje vertical.

rotarI : rota la curva 90 grados a izquierda.

rotarD : rota la curva 90 grados a derecha.

stang : retorna el ángulo de comienzo de la curva.

stang: : setea el ángulo de comienzo de la curva al valor recibido.

swang : retorna el ángulo de barrido de la curva.

swang: : setea el ángulo de barrido de la curva al valor recibido.

Rectangulo

Comentario: clase que representa un rectángulo.

Superclase: Componente.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

containsPoint: :redefine este método que testea si el rectángulo contiene al punto recibido.

displayOn: : dibuja el rectángulo en el *graphicsContext* recibido.

rotarI : rota el rectángulo 90 grados a izquierda.

rotarD : rota el rectángulo 90 grados a derecha.

Elipse

Comentario: clase que representa una elipse.

Superclase: Componente.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

containsPoint: :redefine este método que testea si la elipse contiene al punto recibido.

displayOn: : dibuja la elipse en el *graphicsContext* recibido.

rotarI : rota la elipse 90 grados a izquierda.

rotarD : rota la elipse 90 grados a derecha.

MultiLinea

Comentario: clase que representa una multilínea.

Superclase: Componente.

Subclases: Poligono.

Variables de Instancia: vertices (vértices de la multilínea).

Métodos Principales:

initialize : agrega la inicialización de los vértices.

containsPoint: :redefine este método que testea si la multilínea contiene al punto recibido.

displayOn: : dibuja la multilínea en el *graphicsContext* recibido.

flipH : transforma la multilinea según el eje horizontal.

flipV : transforma la multilinea según el eje vertical.

rotarI : rota la multilinea 90 grados a izquierda.

rotarD : rota la multilinea 90 grados a derecha.

vertices : retorna el conjunto de vértices de la multilinea.

vertices: : setea los vértice al conjunto de puntos recibidos.

addVertice : agrega un vértice a la multilinea.

Imagen

Comentario: clase que representa una imagen.

Superclase: Componente.

Subclases: ImagenAtrib.

Variables de Instancia: *imagen* (contiene el mapa de bits de la imagen).

Métodos Principales:

initialize: : agrega la inicialización de la imagen.

containsPoint: :redefine este método que testea si la imagen contiene al punto recibido.

displayOn: : dibuja la imagen en el *graphicsContext* recibido.

imagen : retorna la imagen.

imagen: : setea la imagen al valor recibido.

Poligono

Comentario: clase que representa un polígono irregular.

Superclase: MultiLinea.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

containsPoint: :redefine este método que testea si el polígono contiene al punto recibido.

displayOn: : dibuja el polígono en el *graphicsContext* recibido.

Texto

Comentario: clase que representa un cuadro de texto.

Superclase: TextEditorView.

Subclases: Field y NodoGrafico.

Variables de Instancia: *origen* (punto origen de cuadro de texto), *corner* (esquina opuesta del cuadro de texto) y *ancho* (ancho de las línea del cuadro de texto)

Métodos Principales:

initialize: : inicializa la figura con los colores y ancho de las paletas correspondientes y el tipo de fuente recibida.

opaco : transforma el componente a opaco.

transparente : transforma el componente a transparente.

cursiva : retorna un valor booleano según el texto esté en cursiva o no.

cursiva: : setea el texto a cursiva o no según el valor booleano recibido.

negrita : retorna un valor booleano según el texto esté en negrita o no.

negrita: : setea el texto a negrita o no según el valor booleano recibido.

subraya : retorna un valor booleano según el texto esté subrayado o no.

subraya: : setea el texto a subrayado o no según el valor booleano recibido.

fuente: : setea el texto al tipo de fuente recibido.

alinear: : alinea el texto según el valor recibido.

containsPoint: :testea si el cuadro de texto contiene al punto recibido.

origen : retorna el origen del cuadro de texto.

origen: : setea el origen del cuadro de texto al punto recibido.

corner : retorna el corner del cuadro de texto.

corner: : setea el corner del cuadro de texto al punto recibido.

ancho : retorna el ancho de línea del cuadro de texto.

ancho: : setea el ancho de línea del cuadro de texto al valor recibido.

foreColor : retorna el color de frente del cuadro de texto.

foreColor: : setea el color de frente del cuadro de texto al color recibido.

backColor : retorna el color de fondo del cuadro de texto.

backColor: : setea el color de fondo del cuadro de texto al color recibido.

Field

Comentario: clase que representa un atributo de tipo *string*, *fecha* o *número* en la etapa de edición de perspectivas.

Superclase: Texto.

Subclases: FieldString.

Variables de Instancia: *atributo* (atributo string, número o fecha con el cual está relacionado) y *objeto* (instancia particular con la cual está relacionada).

Métodos Principales:

displayOn: : dibuja el field en el *graphicsContext* recibido.

atributo : retorna el atributo del field.

atributo: : setea el atributo al valor recibido y actualiza el texto del field.

objeto : retorna el objeto del field.

objeto: : setea el objeto al valor recibido.

FieldString

Comentario: clase que representa un atributo de tipo *string*, en la etapa de autoría.

Superclase: Field.

Subclases: FieldNumero, FieldFecha.

Variables de Instancia: ninguna.

Métodos Principales

atributo: : setea el atributo al valor recibido pero no actualiza el texto del field.

FieldNumero

Comentario: clase que representa un atributo de tipo *número*, en la etapa de autoría.

Superclase: FieldString.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales

defaultControllerClass : devuelve la clase controller para este field: TextoNumeroController.

FieldFecha

Comentario: clase que representa un atributo de tipo *fecha*, en la etapa de autoría.

Superclase: FieldString.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales

defaultControllerClass : devuelve la clase controller para este field: TextoFechaController.

ImagenAtrib

Comentario: clase que representa un atributo de tipo *gráfico*.

Superclase: Imagen.

Subclases: ImagenAtrib.

Variables de Instancia: *atributo* (atributo gráfico con el cual está relacionado) y *objeto* (instancia particular con la cual está relacionada).

Métodos Principales:

displayOn: : dibuja la imagen en el *graphicsContext* recibido.

atributo : retorna el atributo de la imagen.

atributo: : setea el atributo al valor recibido.

objeto : retorna el objeto de la imagen.

objeto: : setea el objeto al valor recibido.

Boton

Comentario: clase que representa un botón.

Superclase: ActionButtonView.

Subclases: ninguna.

Variables de Instancia: *origen* (punto origen del botón), *corner* (esquina opuesta del botón), *atributo* (atributo anchor con el cual está relacionado), *objeto* (instancia particular con la cual está relacionada) y *destino* (nodo destino al cual se navega al presionarlo).

Métodos Principales:

initialize: : inicializa el botón con los colores y ancho de las paletas correspondientes y el tipo de fuente recibida.

opaco : transforma el botón a opaco.

transparente : transforma el botón a transparente.

cursiva : retorna un valor booleano según el texto del botón esté en cursiva o no.

cursiva: : setea el texto del botón a cursiva o no según el valor booleano recibido.

negrita : retorna un valor booleano según el texto del botón esté en negrita o no.

negrita: : setea el texto del botón a negrita o no según el valor booleano recibido.

subraya : retorna un valor booleano según el texto del botón esté subrayado o no.

subraya: : setea el texto del botón a subrayado o no según el valor booleano recibido.

fuente: : setea el texto del botón al tipo de fuente recibido.

alinear: : alinea el texto del botón según el valor recibido.

containsPoint: :testea si el botón contiene al punto recibido.

origen : retorna el origen del botón.

origen: : setea el origen del botón al punto recibido.

corner : retorna el corner del botón.

corner: : setea el corner del botón al punto recibido.

ancho : retorna el ancho de línea del cuadro de texto.

ancho: : setea el ancho de línea del botón al valor recibido.

foreColor : retorna el color de frente del botón.

foreColor: : setea el color de frente del botón al color recibido.

backColor : retorna el color de fondo del botón.

backColor: : setea el color de fondo del botón al color recibido.

atributo : retorna el atributo del botón.

atributo: : setea el atributo al valor recibido.

objeto : retorna el objeto del botón.

objeto: : setea el objeto al valor recibido.

destino : retorna el destino del botón.

destino: : setea el destino del botón al valor recibido.

NodoGrafico

Comentario: clase que representa un nodo en el editor de esquemas.

Superclase: Texto.

Subclases: AccesoGráfico.

Variables de Instancia: *subsistema* (subsistema al que pertenece el nodo) y *links* (conjunto de links que salen y llegan al nodo).

Métodos Principales:

initialize : agrega la inicialización de sus variables de instancia.

subsistema : retorna el subsistema al que pertenece el nodo.

subsistema: : setea el subsistema del nodo al valor recibido.

links : retorna el conjunto de links del nodo.

links: : setea el conjunto de links recibido como los links del nodo.

addLink: : agrega el link recibido al conjunto de links del nodo.

AccesoGrafico

Comentario: clase que representa una estructura de acceso en el editor de esquemas.

Superclase: NodoGrafico.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

displayOn: : dibuja la estructura de acceso en el *graphicsContext* recibido.

LinkGrafico

Comentario: clase abstracta que reúne el comportamiento común de los distintos tipos de links del grafo de esquemas.

Superclase: Linea.

Subclases: EsUn, Relacion.

Variables de Instancia: *fuente* (nodo gráfico que es el fuente del link) y *destino* (nodo gráfico que es el destino del link).

Métodos Principales:

initialize : redefine el método seteando los colores, el ancho y el tipo de fuente a valores predeterminados.

fuente : retorna el nodo gráfico fuente del link.

fuente: : setea el nodo gráfico fuente del link al valor recibido.

destino : retorna el nodo gráfico destino del link..

destino: : setea el nodo gráfico destino del link al valor recibido.

EsUn

Comentario: clase que representa la relación es-un del grafo de esquemas.

Superclase: LinkGrafico.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

containsPoint: : determina si el punto recibido pertenece al objeto.

displayOn: : dibuja el link es-un en el *graphicsContext* recibido.

Relacion

Comentario: clase que representa un link del grafo de esquemas.

Superclase: LinkGrafico.

Subclases: Reflexivo.

Variables de Instancia: *nombre* (texto que contiene el nombre del link).

Métodos Principales:

nombre : retorna el nombre del link.

nombre: : setea el nombre del link al valor recibido.

displayOn: : dibuja el link en el *graphicsContext* recibido.

Reflexivo

Comentario: clase que representa un link reflexivo del grafo de esquemas.

Superclase: Relacion.

Subclases: ninguna.

Variables de Instancia: ninguna.

Métodos Principales:

displayOn: : dibuja el link reflexivo en el *graphicsContext* recibido.

Fuentes

Comentario: cuadro de diálogo que permite setear el tipo, tamaño y estilo de fuente de un objeto con texto.

Superclase: ApplicationModel.

Subclases: ninguna.

Variables de Instancia: *fuentes* (colección de fuentes disponibles), *tamaños* (colección de tamaños disponibles para la fuente seleccionada), *fuente* (fuente seleccionada), *tamaño* (tamaño seleccionado), *negrita* (indica si la fuente va en negrita), *cursiva* (indica si la fuente va en cursiva) y *subraya* (indica si la fuente va subrayada).

Métodos Principales:

openIn: : abre el cuadro de diálogo, seteando como modelo el parámetro recibido.

aceptar : le comunica al modelo el nuevo tipo de fuente y cierra el cuadro de diálogo.

aplicar : le comunica al modelo el nuevo tipo de fuente.

cancelar : cierra el cuadro de diálogo.

PaletaColor

Comentario: Ventana que permite establecer o modificar los colores de los objetos.

Superclase: ApplicationModel.

Subclases: ninguna.

Variables de Instancia: *backColor* (indica cuál es el color de fondo seleccionado), *foreColor* (indica cuál es el color de frente seleccionado) y *backFore* (indica si el color que se va a seleccionar es para el fondo o para el frente).

Métodos Principales:

openEn: : muestra la ventana, seteando como modelo el parámetro recibido.

cambioColor:en: : cambia en la ventana el color seleccionado y le avisa al modelo del cambio.

cambioBackFore : actualiza la ventana ante un cambio de color de fondo a color de frente o viceversa.

foreColor : retorna el color de frente actual.

backColor : retorna el color de fondo actual.

foreColor: : setea el color de frente al parámetro recibido.

backColor: : setea el color de fondo al parámetro recibido.

PaletaLinea

Comentario: Ventana que permite establecer o modificar el ancho de línea de los objetos.

Superclase: ApplicationModel.

Subclases: ninguna.

Variables de Instancia: *ancho* (indica cuál es el ancho de línea seleccionado).

Métodos Principales:

openEn: : muestra la ventana, seteando como modelo el parámetro recibido.

cambioLinea:en: : cambia en la ventana el ancho de línea seleccionado y le avisa al modelo del cambio.

ancho : retorna el ancho de línea actual.

ancho: : setea el ancho de línea al parámetro recibido.

PredicadoEditor

Comentario: Editor que permite establecer un predicado para una estructura de acceso. El predicado es de la forma: <atributo> <operador> <constante>.

Superclase: ApplicationModel.

Subclases: ninguna.

Variables de Instancia: *lisAtrib* (lista de atributos posibles), *atributo* (indica cuál es el atributo seleccionado), *operador* (indica cuál es el operador seleccionado) y *constante* (indica cuál es la constante).

Métodos Principales:

initialize: : abre el editor de predicados vacío, recibe la lista de atributos posibles.

initialize:con: : abre el editor de predicados con el predicado recibido.

aceptar : le comunica al editor de estructuras de acceso cuál es el nuevo predicado y se cierra.

cancelar : cierra el editor sin producir modificaciones.

RelacionarDialog

Comentario: Interface que permite establecer las relaciones entre las clases navegacionales y las clases conceptuales.

Superclase: ApplicationModel.

Subclases: ninguna.

Variables de Instancia: *lisAtrib* (lista de atributos de la clase navegacional), *lisClas* (lista de clases conceptuales del modelo), *lisVar* (lista de variables de instancias de la clase conceptual) y *relaciones* (lista de relaciones ya establecidas).

Métodos Principales:

open : abre la interface con las relaciones ya establecidas si las hay.

relacionar : establece la relación entre los distintos componentes seleccionados de las listas (de atributos, de clases y de variables de instancia), pidiendo cuál es el método de acceso a la variable de instancia.

borrar : elimina una relación ya establecida.

aceptar : le comunica al editor de nodos cuales son las nuevas relaciones y se cierra.

cancelar : cierra la interface sin producir modificaciones.

SelectNodoDialog

Comentario: Cuadro de diálogo para agregar un nodo a una hiperbase.

Superclase: SimpleDialog.

Subclases: ninguna.

Variables de Instancia: *lisClases* (lista de nodos navegacionales), *lisPerps* (lista de perspectivas del nodo navegacional seleccionado), *lisRel* (lista de relaciones del nodo navegacional seleccionado) y *listInst* (lista de instancias correspondientes a la relación seleccionada) y *conexiones* (lista de conexiones entre relaciones e instancias).

Métodos Principales:

initialize : inicializa el cuadro de diálogo con la lista de nodos navegacionales disponibles.

conectar : establece la conexión entre la relación definida previamente y la instancia particular para el nodo que se está agregando.

aceptar : le comunica al editor de hiperbase el nodo, la perspectiva y las conexiones con las instancias del modelo conceptual del nodo que se debe agregar.

cancelar : cierra la interface sin producir modificaciones.