Proyecto de Taller V:

Un CASE para OLAP

- 1999 -

Tutor: Ing. Fernando Carpani

Integrantes del Grupo:
Alejandro Picerno (C.I. 2.015.340-3)
Mariana Fontán (C.I. 2.854.303-6)

Indice

Un case para OLAP

1. Introducción

1.1. El proyecto

Este proyecto de Taller V, es parte de la creación de una herramienta CASE para el diseño de sistemas basados en aplicaciones OLAP. La herramienta CASE propuesta, se basa en la creación y explotación de un repositorio que contiene esquemas multidimensionales.

El trabajo se basa en el modelo conceptual CMDM. Este modelo de datos para la especificación de bases de datos multidimensionales, cuenta con una representación gráfica para ser utilizada por el diseñador y una representación abstracta para realizar chequeos automáticos de consistencia o ambigüedad. El lenguaje gráfico permite expresar la estructura de los datos y el comportamiento frente a determinadas situaciones (mediante restricciones).

1.2. Objetivo

El objetivo de este proyecto es la construcción de una herramienta gráfica para editar esquemas multidimensionales. Además, dicho editor debe ser multiplataforma y suficientemente flexible como para permitir incorporar nuevas herramientas.

Dado el requerimiento de extensibilidad, se construyó un editor gráfico no sólo extensible sino genérico, que edita esquemas multidimensionales como caso particular. Este editor permite ser configurado para un modelo particular, manipularlo y aplicarle métodos genéricos y métodos particulares de éste.

1.3. Organización del informe

El informe se organiza de la siguiente manera: en el capitulo a continuación se presenta el análisis y los requerimientos, luego el diseño y la implementación del prototipo y finalmente la instancia del editor para CMDM. En los anexos, se presentan documentos realizados en el transcurso del proyecto y otros que tratan temas que se mencionan en este informe pero desarrollados más en detalle.

2. Análisis

En esta sección primero se presentan los requerimientos y luego se divide el análisis en dos secciones: el análisis del modelo y el análisis del editor.

La sección del análisis del modelo, pretende reflejar la especificación del lenguaje gráfico del modelo CMDM. Identificamos los objetos que intervienen en el modelado de los esquemas multidimensionales pero sin tener en cuenta la edición del modelo.

La sección de análisis del editor, se concentra en el editor propiamente, como pueden ser los requerimientos de facilidades de uso, la extensibilidad, etc.

2.1. Requerimientos

Se debía construir una herramienta gráfica para editar esquemas multidimensionales basándose en el lenguaje gráfico del modelo CMDM.

El editor debería tener algún mecanismo para comunicarse con el repositorio pudiendo crear o recuperar esquemas del repositorio y trabajar con varios esquemas a la vez (al menos dos) para permitir al usuario, al menos, una forma rudimentaria de reutilización en función de operaciones de cortar y pegar.

Dado que el repositorio ya puede soportar varias plataformas, era deseable que el editor también presentara esta característica.

Se debía tener en cuenta la extensibilidad del editor. Esto es, por ejemplo, poder agregar un nuevo tipo de componente. Debía estar claramente documentado el mecanismo para el agregado de extensiones al producto.

Además, se tenían ciertos requerimientos en cuanto a facilidades de uso que mencionamos a continuación:

El editor debería ser cómodo y fácil de usar. Por lo tanto, debería proveer facilidades como por ejemplo copiar y pegar y menúes según contexto.

Se debería permitir la definición de un componente nuevo dentro de la edición del objeto que lo contenga. Por ejemplo, al estar editando una dimensión, poder definir un nivel nuevo sin salir del ambiente donde se está. Además, esto tendría que poder hacerse de forma ordenada para que el usuario “no se pierda”.

2.2. Contexto de trabajo

Dado que era deseable que el editor fuese multiplataforma, y el repositorio había sido desarrollado en Java, se optó por utilizar Java como lenguaje de programación.

Se utilizó UML para modelar y documentar el proyecto. Para la programación del prototipo se usó fundamentalmente la herramienta JBuilder2 de Borland y VisualJ de Microsoft sobre plataformas Windows98 y Windows NT Server. El prototipo además fue probado en plataformas Unix.

2.3. Análisis del modelo

Esta sección, se concentra en el análisis basado en el lenguaje gráfico del modelo CMDM (ver [CMDM]), describiendo los objetos que deberá editar el editor.

Se identifican los objetos que surgen directamente del modelo y sus relaciones estáticas, construyendo un diagrama de clases. De los distintos escenarios que surgen de las operaciones para la definición de un esquema, se identifican las operaciones básicas de las clases (ver anexo: Análisis detallado).

2.3.1. CMDM. Lenguaje gráfico.

La idea del lenguaje es representar todos los elementos posibles de un esquema multidimensional. El lenguaje entonces, deberá ser capaz de expresar:

- Especificación de esquemas de Dimensiones y Niveles.
- Especificación de esquemas de Cubos.

La especificación de los cubos debe tener en cuenta la existencia o no de la dimensionalidad genérica.

Para apoyar el proceso de desarrollo de los sistemas, se provee una construcción que es capaz de especificar un “tipo” de cubo.

Se asume que cualquier construcción de las anteriores tiene asociado un conjunto de condiciones a las que llamaremos restricciones. Estas restricciones se especificarán mediante “macros gráficas” o directamente en texto.

Niveles

Un nivel de agregación, no tiene una diferencia sustancial con un conjunto de entidades del Modelo Entidad-Relación. Es un nombre con un determinado dominio asociado. Este dominio puede tener una estructura tan compleja como sea necesario para especificar lo que se desea.

De esta forma, un dominio se puede especificar de una forma similar a los conjuntos de entidades.

Cada nivel no tiene porqué pertenecer a una única dimensión, sino que puede pertenecer a varias dimensiones simultáneamente.

Si un atributo de un nivel aparece subrayado, se asume que ese atributo es identificador dentro del nivel.

Para ejemplificar, tomemos el caso de un esquema para el análisis de las ventas de una empresa automotora. Entonces, se podría tener un nivel que fuese vendedor, en el cual se tendrían datos del vendedor como nombre, código, etc. El identificador del nivel podría ser el código del vendedor. Además se podría tener otro nivel que fuese sucursal, otro nivel podría ser modelos de autos, y así ser irían identificando todos los niveles que interesen.

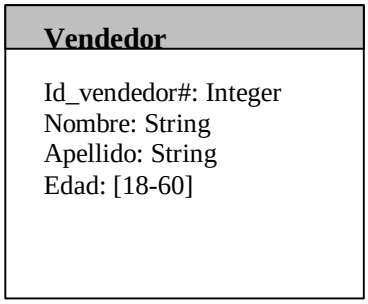

Fig. 1. Nivel vendedor

Dimensiones

Una dimensión, está formada por un conjunto de niveles organizados en una estructura jerárquica que especifica las posibles formas de agregación. Por esto, la especificación de una dimensión estará determinada fundamentalmente por el grafo que representa la estructuración jerárquica de los niveles.

Un punto a resaltar es que se asume la debilidad de un nivel con respecto al superior. Esto es que los atributos subrayados, son identificadores pero con respecto al mismo elemento del nivel superior, excepto para los niveles más altos en una jerarquía.

En caso que un atributo de un nivel inferior, o sea que no es el más alto de una jerarquía, termine con el carácter #, se asume que ese atributo identifica los elementos del nivel independientemente del nivel superior. Si el nivel es el más alto de la jerarquía, entonces es lo mismo subrayar que agregar el numeral.

En el ejemplo de la automotora, se podría tener una dimensión vendedores que fuese una jerarquización de los vendedores por sucursal, por ciudad, etc. Otra para los autos, dividiendolos en color, modelo, etc. También se podría tener una jerarquia de fecha, siendo un nivel día, un nivel mes, uno semestre, trimestre, año, etc, para poder analizar la información por estos distintos períodos.

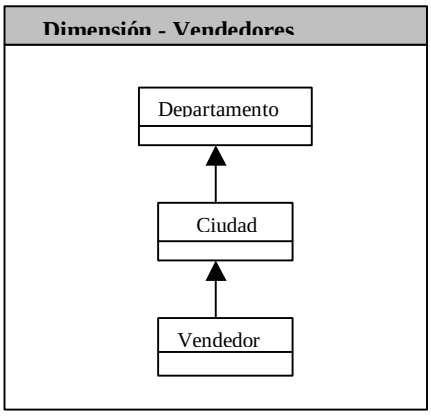

Fig. 2. Dimensión vendedores

Relaciones dimensionales

Una relación dimensional refleja una relación entre dimensiones.

De esta forma, se indica que se desea que en el modelo considerado debe existir al menos un cubo en el que participen esas dimensiones. En caso que exista una punta de flecha sobre una dimensión, solamente se consideran cubos que tengan algún nivel de esa dimensión como medida explícita (sin generalidad dimensional).

Siguiendo el ejemplo, se podría tener una relación dimensional para la información de las ventas. En esta relación se podrían tener las dimensiones que interesen para el análisis de las ventas como Vendedores, Autos, cantidades vendidas.

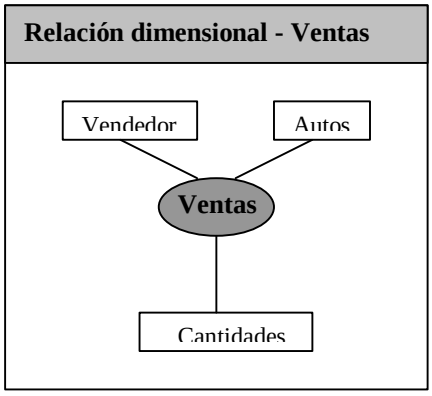

Fig. 3. Relación dimensional Ventas

Cubos

Un cubo está formado por los niveles de las dimensiones que corresponde y una medida que también es un nivel de determinada dimensión.

Una punta de flecha sobre un nivel indica que ese nivel cumple el rol de medida en ese cubo. Para reflejar dimensionalidad genérica en ese cubo, se debería eliminar esa punta de flecha.

En el ejemplo, se podría tener un cubo que fuese ventas de automóviles donde se tuviesen los niveles modelo de auto, color de auto, cantidad vendida y vendedor, siendo la cantidad vendida la medida.

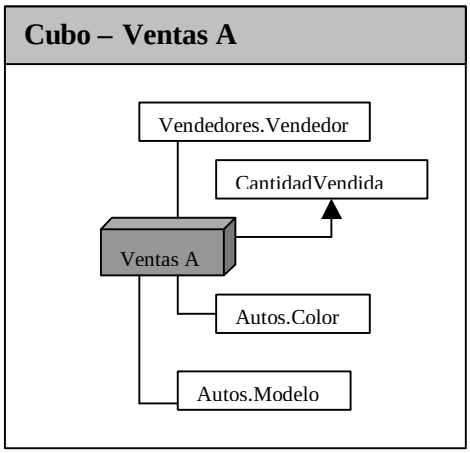

Fig. 4. Un cubo para ventas.

Para proveer ciertas facilidades para la edición de un esquema, se han hecho algunas extensiones al modelo CMDM. Aparecen los conceptos de jerarquías, restricciones standard y componentes pendientes que se detallarán a continuación.

Jerarquías

Se deberá poder definir jerarquías de niveles por fuera de las dimensiones, o sea, existirán independientemente de las dimensiones.

Estas jerarquías tendrán la siguiente utilidad:

- podrán ser reutilizadas en varias dimensiones, es decir, facilitaran el diseño cuando aparezca la misma jerarquía en más de una dimensión. Las dimensiones, por lo tanto podrán tener no solo niveles sino también jerarquías asociadas.
- servirán para simplificar el gráfico del diseño de la dimensión cuando se quiera ver la dimensión desde alguna visión particular, pudiendo “agrupar” niveles con algún criterio dentro de una jerarquía. Se debería poder seleccionar un grupo de componentes de la jerarquía y compactarlas en otra componente, o dada una componente comprimida descomprimirla. A nivel gráfico, los componentes compactados, se verían como un sólo componente y diferenciado de alguna forma. Al pedir descompactarlo, se volverían a ver los componentes originales.

También, sería deseable que las jerarquías fuesen parametrizables, es decir, al usarlas en una dimensión, poder pasar algunos parámetros como el nombre de algún nivel, tipo de algún nivel, etc.

Como ejemplo, se podría tener una jerarquía Fecha en lugar de una dimensión y así reutlizarla en varias dimensiones.

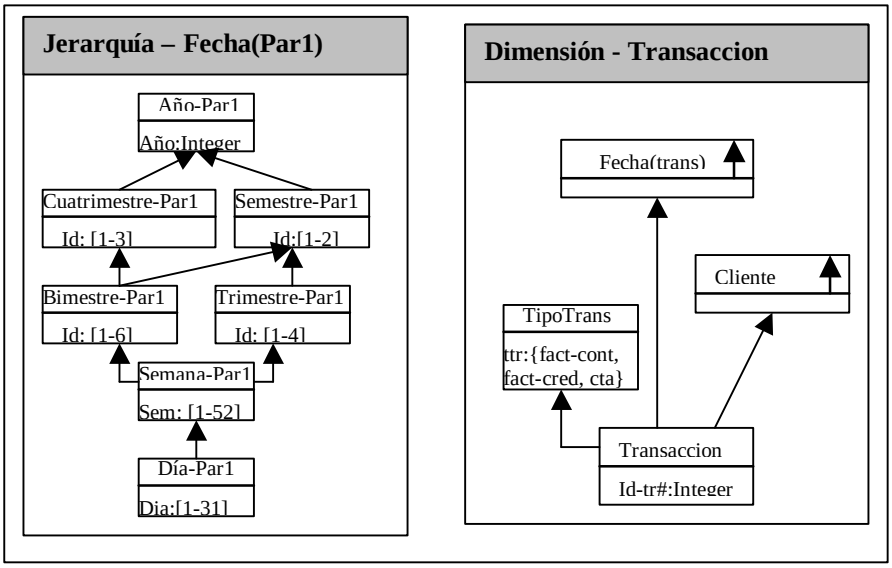

Fig. 5. Jerarquía Fecha con parámetro y su uso en una dimensión.

Pendientes

Se deberá poder marcar las componentes de un esquema como pendientes. Es decir, que se indique de alguna forma que su diseño no está completo y luego el usuario sepa qué está pendiente.

La utilidad de esto es que el diseñador pueda ir armando el esquema de una forma flexible, posponiendo la definición de los detalles para cuando lo considere conveniente. Se hace necesario además una utilidad que le permita recordar aquello que dejó sin terminar.

Una posibilidad planteada es que se dé la opción a que el editor marque todo como pendiente y el diseñador irá desmarcando las componentes a medida que las considere completas

Restricciones standard

En principio, habrá un lenguaje para las restricciones, pero la idea es simplificar la definición de las restricciones en los casos más comunes.

Las restricciones standard, serán macros sobre el lenguaje de las restricciones que facilitarán la definición de éstas en varios casos. Incluso podrían tener asociada una representación gráfica.

Algunas restricciones standard podrían ser:

- Asociaciones entre niveles como categorizaciones, relaciones, etc.

- Multiplicidades.
- Definir cual nivel va a ser medida en un cubo o relación dimensional.
- Restricciones para drill up.

2.3.2. Diagrama de clases identificadas para el modelo.

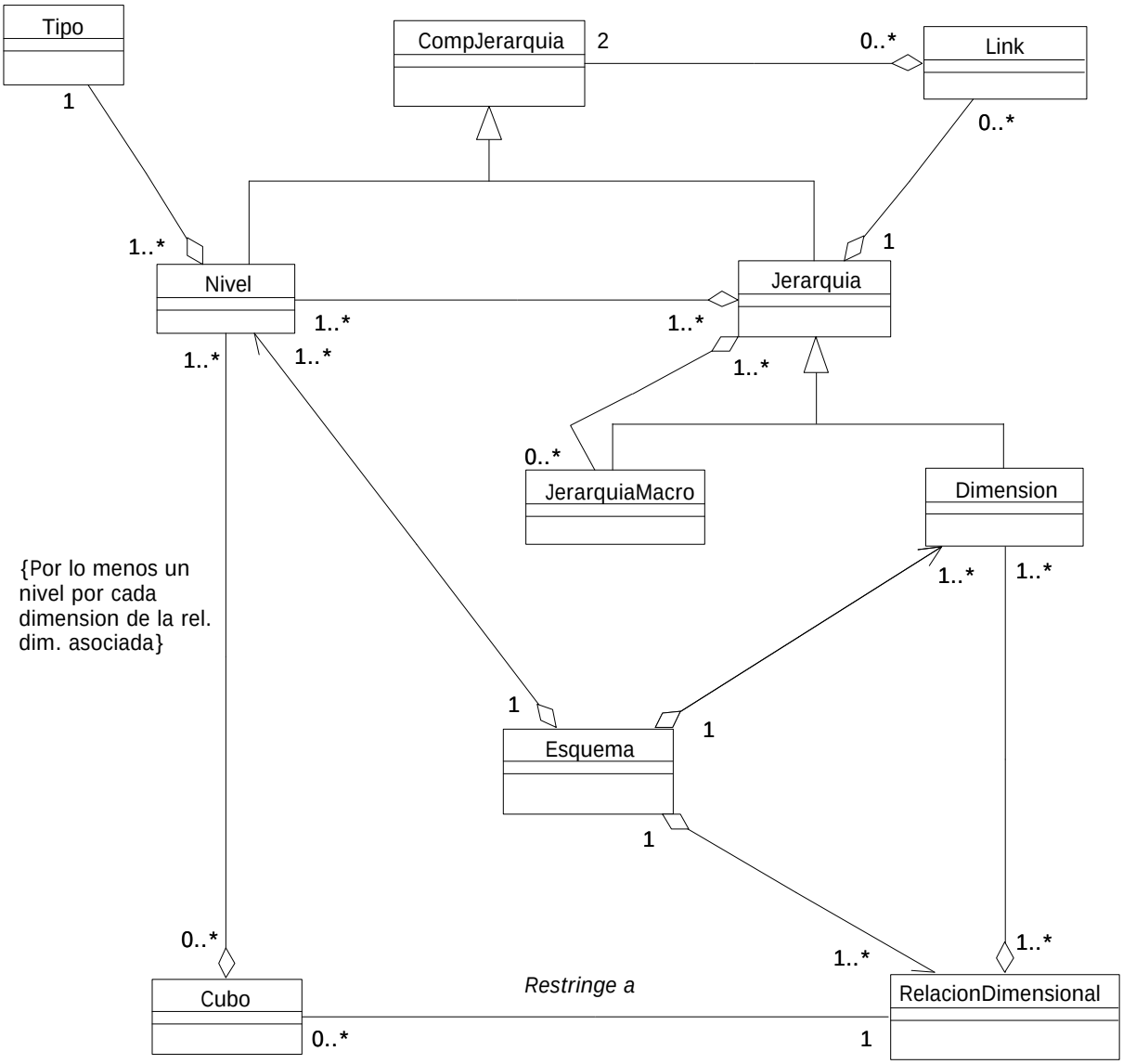

Fig. 6. Diagrama de clases para el modelo CMDM

Se tienen clases que se desprenden del análisis del lenguaje gráfico del CMDM. Estas son claramente las que representan a los objetos identificados como Esquema, Nivel, Dimensión, etc. Además, aparecen clases para poder soportar los requerimientos de facilidades de uso como jerarquía.

Una jerarquía, en principio, sería un conjunto de niveles y links entre niveles. Por lo tanto, las dimensiones podrían verse como una jerarquía, entonces, se consideran a las dimensiones como una subclase de la nueva clase jerarquía. Como una de las finalidades de las jerarquías es su reutilización en varias dimensiones, esto implica que las jerarquías pueden estar formadas no solo por niveles y links entre ellos, sino también por otras jerarquías. Es así, que ahora los links podrán ser no sólo entre niveles sino también entre jerarquías o entre nivel y jerarquía. Para ello es que se agrega una superclase de nivel y jerarquía: CompJerarquica, que va a ser componente general de las jerarquías, y los links serán entre objetos de esta clase. También para la reutilización de jerarquías, aparece la clase JerarquíaMacro que es una jerarquía que puede ser agregada dentro de otra jerarquía como en una dimensión.

También hay clases que surgen de las abstracciones luego de considerar los métodos de las clases. Estas son CompConRestriccion y CompPersistente.

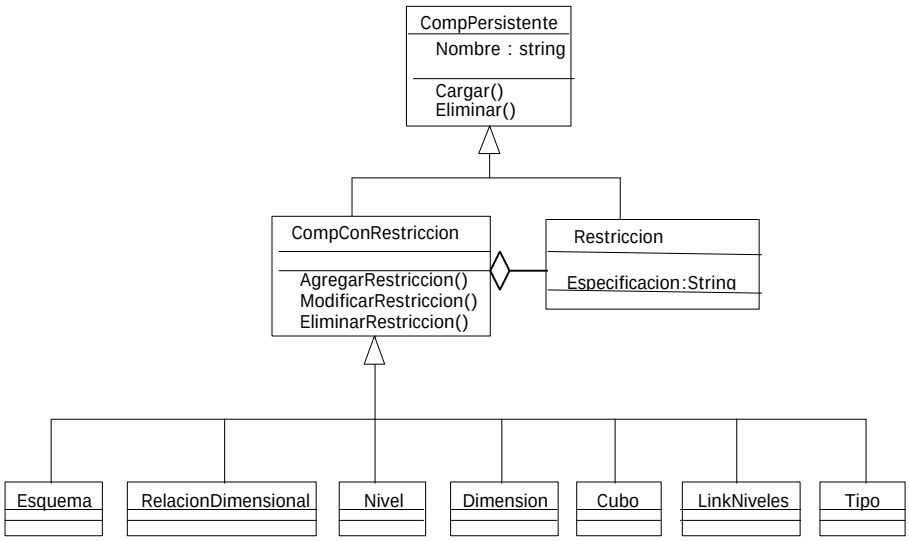

Fig. 7. Diagrama de clases para los componentes que surgieron de las abstracciones

2.3.3. Consideraciones sobre algunas relaciones del diagrama de clases

La clase restricción está asociada a todos los elementos del modelo que son subclase de ComponenteConRestriccion: nivel, jerarquía, dimensión, relación dimensional, cubo, tipo, esquema, ya que las restricciones existen para un objeto determinado.

Las jerarquías macro podrán relacionarse con cualquier jerarquía macro excepto ella misma.

Los niveles relacionados con un cubo, deberán ser por lo menos uno por cada dimensión dentro de la relación dimensional a la cual restringe el cubo.

2.3.4. Escenarios

Se identificaron los siguientes escenarios para la edición de un esquema multidimensional:

. Crear un esquema

. Eliminar un esquema

. Cargar un esquema

. Guardar un esquema

. Modificar un esquema

  . Agregar un nivel a un esquema

  . Modificar un nivel de un esquema

    . Definir tipo

    . Modificar tipo

  . Eliminar un nivel de un esquema

  . Agregar dimensión a un esquema

  . Modificar una dimensión de un esquema (modificar una jerarquía)

  . Eliminar una dimensión de un esquema

  . Agregar una relación dimensional a un esquema

  . Eliminar una relación dimensional de un esquema

  . Modificar una relación dimensional de un esquema

. Agregar una restricción a una ComponenteConRestriccion

. Conectar dos componentes en una jerarquía

. Desconectar dos componentes de una jerarquía

De estos escenarios surgieron las operaciones básicas de los objetos. En el anexo Análisis Detallado, se extiende este punto, incluyendo los diagramas dinámicos en UML y el detalle de las operaciones identificadas.

2.4. Análisis del editor

Siendo un producto interactivo, con interfase gráfica, es importante la apariencia y las facilidades de uso que se provean y esto es lo que se trata en esta sección.

El editor tiene una pantalla principal y dentro de ésta se irán abriendo las ventanas para editar los distintos componentes del esquema.

Dentro de la ventana principal se tendrá disponibles las operaciones básicas del editor, como nuevo componente, abrir y guardar, herramientas de edición como copiar, pegar y otras utilidades propias del editor, así como herramientas propias del modelo.

Hay un visualizador que muestra todos los componentes para poder seleccionar fácilmente cual editar y facilitar la navegación del usuario entre los diversos diagramas de los componentes.

Dentro de la ventana de edición de un componente, se verán todos los objetos que componen al objeto editado y se podrán también editar éstos. Por ejemplo, dentro de una dimensión, se verá el diagrama jerárquico de los niveles y/o jerarquías que la componen en forma de grafo y desde allí se podría seleccionar, por ejemplo, un nivel para editarlo. También se podrían estar editando varios objetos a la vez en varias ventanas.

También, para cada diagrama, hay una toolbar con los componentes que pueden agregarse al objeto en edición, o de operaciones que se le puedan aplicar al objeto, etc.

Para modificar las propiedades del objeto editado, como el nombre, estado de pendiente, etc., se tiene un formulario que se abrirá mostrando todas las propiedades y permitiendo modificar sus valores (hoja de propiedades).

De lo anterior, se observa que cada objeto tiene cierta información gráfica sobre todo cuando compone a otro objeto. Esta información incluye posición en la pantalla, tamaño, color, etc. Además, se debe tener distinta información gráfica según qué objeto está componiendo. Esto es, porque en caso de objetos que pueden estar componiendo a varios, no en todas las pantallas tendrán la misma posición y tamaño. Por ejemplo, un nivel que está en varias dimensiones, no estará en la misma posición en la pantalla de edición de todas las dimensiones.

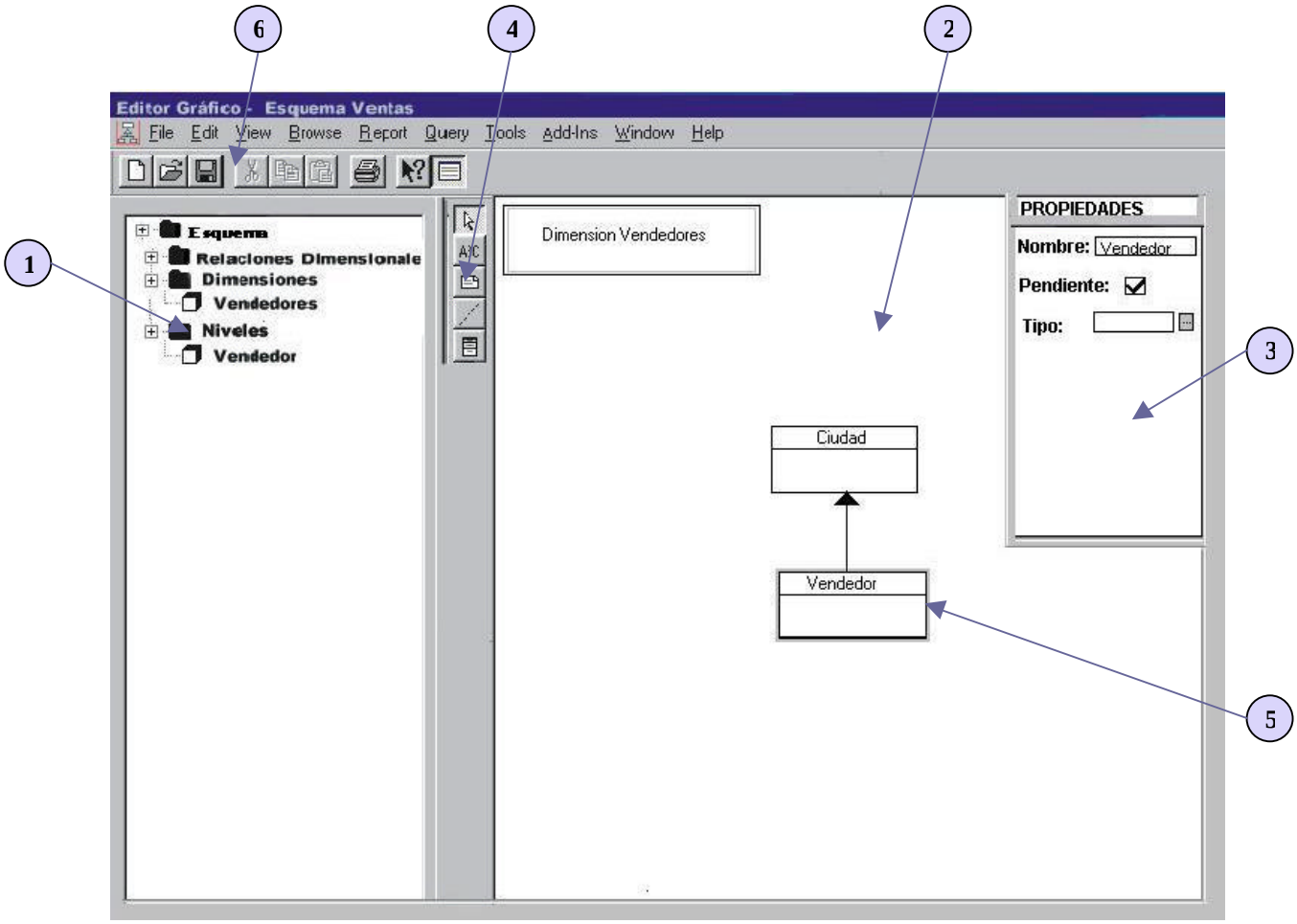

Fig. 8. Diseño de la pantalla principal del editor. 1)Visor de componentes. 2)Ventana de edición de un componente . 3)Hoja de propiedades. 4)Toolbar del diagrama. 5)Componente seleccionado. 6)Toolbar de las herramientas del editor.

2.4.1. Objetos gráficos identificados

De lo anterior, se identificaron ciertos objetos gráficos:

- Ventana principal.
- Ventanas de edición de los objetos.
- Visualizador gráfico para ver todas las componentes.
- Toolbars: una para las funciones generales del editor y otra para la ventana de edición del objeto.
- Formularios para visualizar y modificar propiedades de los objetos (hoja de propiedades).

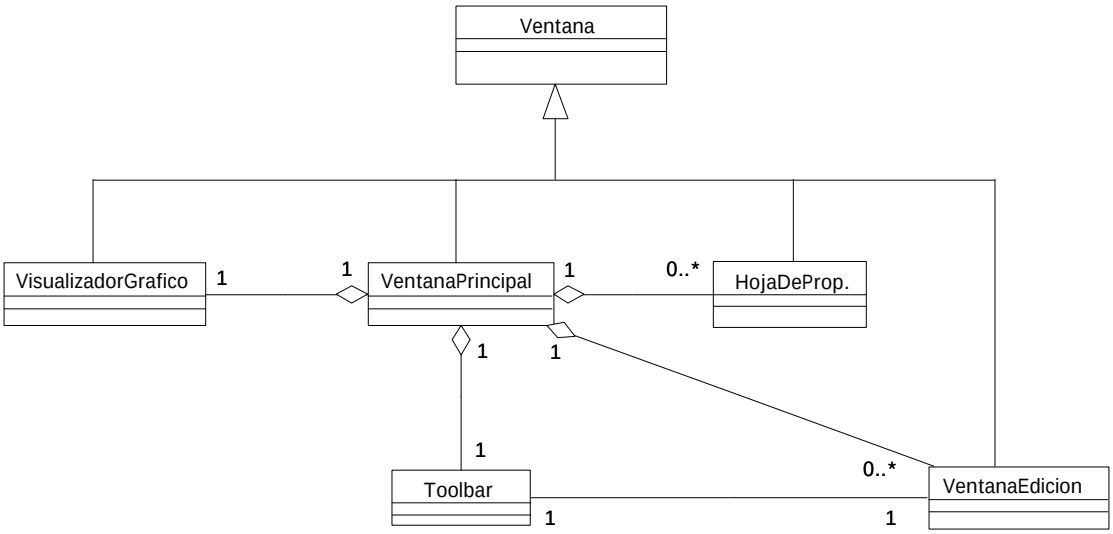

Fig. 9. Diagrama de clases de los objetos gráficos de la pantalla principal del editor

3. Diseño

Dado el fuerte requerimiento de extensibilidad, se comenzó a analizar las diferentes posibles extensiones y yendo al extremo se podría querer agregar o modificar cualquier tipo de componente del modelo o cualquier herramienta. Es así que se comenzó a pensar en la posibilidad de hacer un editor que más que extensible fuese genérico, que en principio no tuviese en cuenta nada particular del modelo a editar.

El editor, está dividido en una parte que contiene todo el manejo de un editor gráfico genérico y una parte con los objetos editables propios del modelo. Entonces, el editor interactúa con los objetos a editar exigiendo que tengan ciertas propiedades y le provean ciertos servicios. Los cambios en el modelo no afectarán a la parte del editor y todas las herramientas propias del modelo serán provistas por éste (como en esquemas multidimensionales el generar código para repositorio, chequeo sintáctico de restricciones, etc.)

El editor, por lo tanto, está diseñado para poder editar cualquier modelo. Se diseñaron objetos que son componentes genéricos, sin tener en cuenta el problema particular del modelo de esquemas multidimensionales. Para usar el editor para editar un nuevo modelo (por ejemplo para tener un editor de MER), se deberá programar las clases que representen los componentes de éste (las entidades, las agregaciones, las relaciones, etc). Se proveen objetos editables básicos, métodos y comportamientos por defecto, requiriendo muy poca programación para un modelo que se ajuste a estos comportamientos, y necesitando mayor programación a medida que el modelo se aleje más de éstos. (ver Anexo: Manual de Instanciación)

Se definirán los patrones y las interfaces que deberán seguir los objetos para ser editables en este editor y luego, se instanciará el editor para el modelo de esquemas multidimensionales.

3.1. Diseño del editor

El módulo que maneja todo el funcionamiento básico del editor genérico está compuesto por la pantalla principal del editor y los demás objetos gráficos que la componen.

Como se había mencionado en el análisis, los objetos gráficos que la componen son:

- Un visualizador de la jerarquía de los componentes editados que muestra los componentes del modelo en una estructura de árbol.
- Una ventana de edición de los diagramas de los componentes junto con la toolbar de los objetos disponibles para agregar según el tipo de diagrama. Se podrán tener abiertas más de una a la vez, minimizar, maximizar e iconizar.
- Una hoja de propiedades para mostrar y editar las propiedades del objeto editado.

- Una barra de menúes.

Por lo tanto, este módulo provee las funcionalidades de:

- Desplegar y manejar la pantalla principal del editor.
- Desplegar el visualizador de componentes editados.
- Desplegar las pantallas de edición de cada diagrama con la toolbar adecuada.
- Desplegar los formularios para editar propiedades.
- Manejar restricciones de operaciones según el contexto
- Manejar y cargar los menues particulares para el modelo a editar.

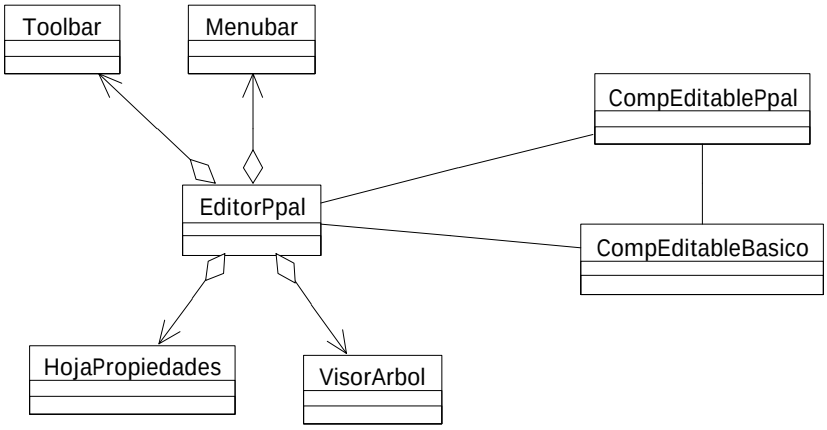

Fig. 10. Diagrama de clases para los componentes básicos del editor.

En el diagrama de clases, se ve cómo están relacionados los distintos componentes del módulo y además como interactúan con lo que será el componente principal del modelo editado y el componente editable básico. Esta relación es a través de servicios que le estará pidiendo el editor a los objetos editables.

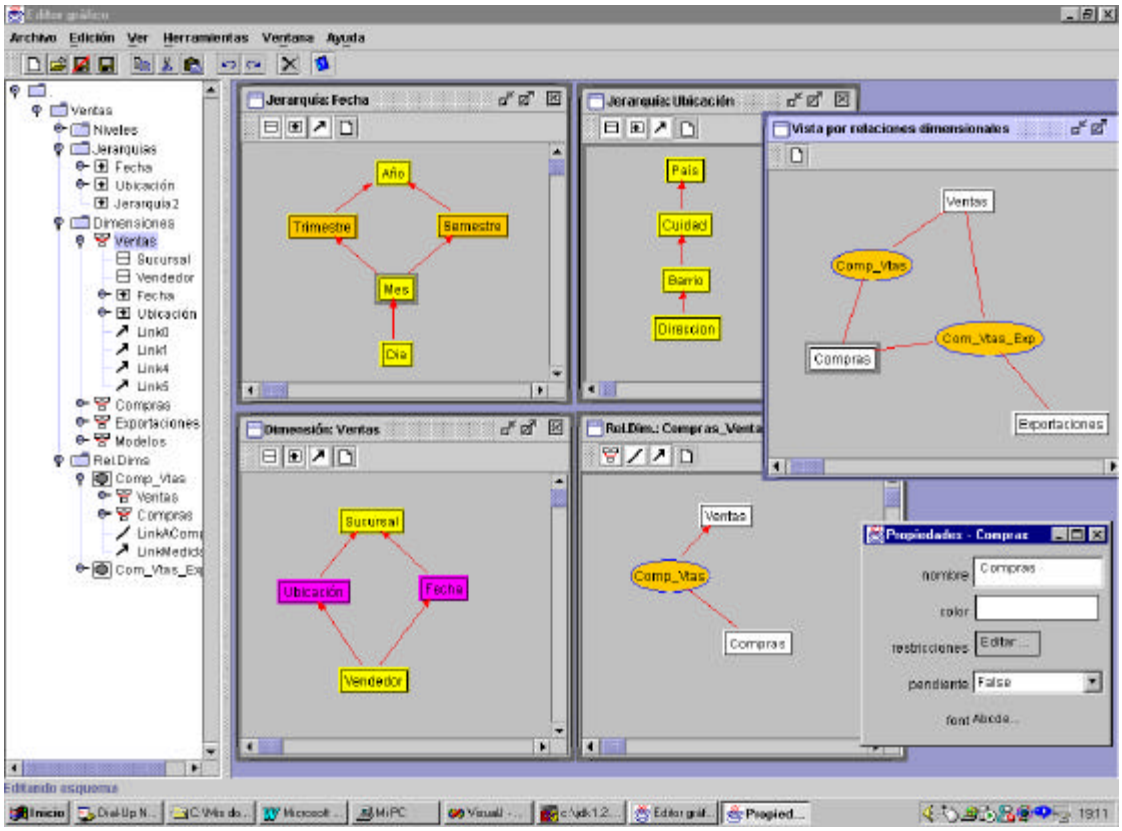

Fig. 11. Pantalla principal del prototipo editando un esquema multidimensional

3.1.1. Clase principal del editor (EditorPpal)

Hay una clase que es la que se encarga de todo el manejo de la pantalla principal y que tiene el control del editor.

Esta clase es un formulario que es la pantalla principal del editor. Tiene una toolbar y una menubar y se divide en paneles donde estarán el visor de componentes y las ventanas de edición de componentes. Maneja los eventos que disparen los menues y los botones de la toolbar. Al recibir un evento de un item de menú, ésta disparará el método adecuado del objeto del modelo que corresponda. Por ejemplo, al hacer un copy, ésta llamará al copy del diagrama en edición, y al entrar a un item del menú que sea una herramienta específica del modelo, ésta llama al método correspondiente propio del modelo.

El menú de la pantalla principal con sus items es el siguiente:

>**Archivo**

>Nuevo

(aquí irán opciones según los componentes que tenga el modelo: nuevo nivel, nueva dimensión, etc.)

Abrir

Guardar

Guardar como

Cerrar

Configuración

Salir

>**Edición**

Deshacer

Seleccionar

Copiar

Cortar

Pegar

>**Ver**

Visor de componentes

Ventana de propiedades

Barra de herramientas

>**Herramientas**

(aquí irán las herramientas que provea el modelo como podría ser generar código para el repositorio, analizador sintáctico de restricciones, etc)

>**Ventana**

(aquí aparecerán las ventanas abiertas)

>**Ayuda**

Nota: los puntos precedidos por “>” corresponden a menues o submenues, los demás a items.

3.1.2. Hoja de propiedades

La hoja de propiedades es un formulario que muestra las propiedades del componente activo.

En caso de estar editando subcomponentes dentro de un diagrama, se mostrarán las propiedades del objeto que tiene el foco dentro de éste, de lo contrario se mostrarán las propiedades del componente propietario del diagrama. Si se están seleccionando varios componentes, también se mostrarán las propiedades del propietario del diagrama.

Este formulario es genérico para todos los componentes. Se usan métodos de introspección, para obtener las propiedades del componente, desplegarlas y llamar a las funciones set y get correspondientes del objeto.

También, se puede programar un formulario hecho especialmente para el objeto. Si se provee éste, sustituirá a la hoja de propiedades.

3.1.3. Visor gráfico de componentes (VisorArbol)

Este es una clase que se encarga de mostrar los componentes del modelo editado en una estructura de árbol. Permitirá ir navegando por las ramas y acceder desde allí a los distintos componentes para su edición. Al seleccionar el componente en el árbol, se muestra su diagrama (en caso de tenerlo) y sus propiedades en la hoja de propiedades.

3.1.4. Configuración del editor

Hay una clase para la configuración del editor. Esta clase es un formulario en el cual se ingresan los datos sobre el componente principal del modelo a editar. También se permite guardar las distintas configuraciones en archivos y leer una configuración desde un archivo.

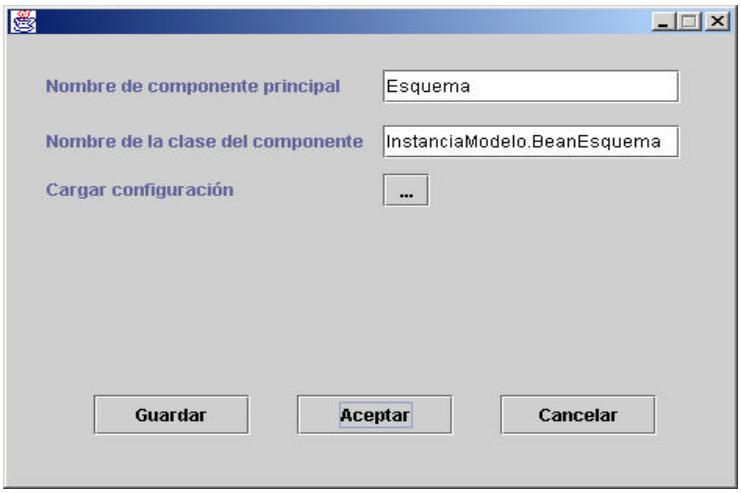

Fig. 12. Formulario del prototipo para configurar el modelo a editar.

3.2. Diseño de los componentes editables básicos

Todo modelo editable deberá tener un *objeto principal* que será el que represente al modelo. Por ejemplo, en el caso de esquemas multidimensionales, éste será el esquema. Este objeto principal tendrá la información de todos los objetos que lo componen y proveerá las herramientas específicas que se implementen para el modelo.

Cada objeto que componga al modelo, incluyendo el componente principal, podrá tener un *diagrama* que contenga subcomponentes. Los componentes pueden ser *globales o locales*. Los globales, serán aquellos que pueden aparecer en varios diagramas pero que representarán al mismo objeto. Estos, en cada diagrama tendrán ciertas propiedades locales propias (color, tamaño, posición) pero compartirán los datos básicos con todas las instancias del mismo (nombre, estado de pendiente, diagrama de componentes). Los componentes locales sólo existirán dentro de un diagrama.

Todo componente editable debe seguir ciertos patrones y proveer ciertos servicios. Los objetos pueden tener subcomponentes y podrán ser subcomponentes de otro. Por lo tanto, estos tendrán algunas o todas de las siguientes características:

- Poder aparecer en la toolbar para algún diagrama y tener representación gráfica en el diagrama.
- Tener un diagrama asociado donde crear la jerarquía de los objetos que lo componen.
- Tener un conjunto de propiedades editables en un formulario.

Entonces, los objetos tendrán las siguientes propiedades:

- Un nombre identificatorio.
- El estado de pendiente.
- Un ícono para mostrar en la toolbar.
- Un diagrama para editar sus componentes.

Y deberán ofrecer los siguientes servicios:

- Guardarse.
- Leerse.
- Mostrarse en un diagrama.
- Moverse en un diagrama.
- Cambiar el tamaño dentro del diagrama.
- Seleccionarse (al tomar el foco o ser seleccionado, cambiar el aspecto)
- Levantar el diagrama con la toolbar adecuada para editar sus componentes.
- Proveer el ícono para la toolbar de su componente padre.
- Proveer el árbol de jerarquía de sus componentes.
- Levantar el formulario con sus propiedades para editar.

Diagrama

La ventana del diagrama que tenga el componente será subclase de un objeto base para los diagramas. Esta clase es un formulario que proporcionrá los siguientes servicios:

- Cargar la toolbar con los componentes adecuados.
- Agregar un componente en una posición del panel.
- Obtener el componente activo (el que tiene el foco)
- Seleccionar uno o más componentes.
- Eliminar el componente que tiene el foco o los componentes seleccionados.
- Copiar o cortar el componente que tiene el foco o los componentes seleccionados.
- Agregar uno o varios componentes del "portapapeles" (aquel o aquellos que se cortaron o copiaron).
- Guardarse. (para guardar la información del diagrama)
- Leerse a partir de la información guardada.

3.2.1. Clases editables básicas

Para que los componentes editables sigan los patrones necesarios y provean los métodos básicos que se exigen para el editor, se eligió que esto fuese a través del mecanismo de herencia.

Las clases que se detallarán en esta sección son aquellas que los componentes editables para cualquier modelo tendrán que extender.

Estas clases principales son:

- El componente básico para un objeto editable principal (aquel que tendrá el manejo de todo el modelo) Esta clase deberá proveer servicios que le exigirá el editor.
- El componente para un objeto editable básico (este será un componente del modelo pero no podrá ser editable directamente, es decir, no sino a través de un objeto editable principal)
- Una diagrama básico, que será la clase base para editar cualquier nuevo diagrama.

Clase para componente editable básico principal.

La clase para el componente editable principal, es una extensión de la de componente editable básico agregando ciertos servicios que proveerá a la clase principal del editor.

Dentro de los servicios que deberá proveer al editor se encuentran: dar una lista de las herramientas que provee el modelo (como ejemplo, generar código para el repositorio, etc.), dar una lista de los componentes que tiene el modelo (nivel, dimensión, rel. dimensional, etc), y proveer métodos como abrir, guardar, crear nuevo componente de tal tipo (que nos proveyó), ejecutar una herramienta propia del modelo (de las que proveyó) .

Clase para componente editable básico.

La clase para el componente editable básico tiene un conjunto de propiedades gráficas y un conjunto de propiedades que les llamaremos datos base. En el caso de que el objeto pueda aparecer como componente de varios objetos, todos las "apariciones" del objeto podrán tener distinta información gráfica como color, tamaño, etc, y tendrá la misma información base como nombre, estado de pendiente o no, etc. Además, otra información que será única para un objeto es el diagrama de componentes. Por lo tanto, al agregar un componente a un diagrama, se crea un nuevo objeto para representarlo en ese diagrama, pero si es un componente existente, éste deberá compartir los datos base y su diagrama con todos los objetos que estén representando al mismo objeto que él.

Todo componente editable deberá proveer su ícono que se usará en las toolbars y en el nodo del visor de componentes. Además deberá indicar las propiedades que permitirá editar (Seguramente no se desee mostrar y permitir editar todas las propiedades del objeto).

Clase para el diagrama básico

El diagrama básico será la superclase de todos los diagramas de componentes que haya en el modelo a editar.

Esta tiene los componentes básicos del diagrama y provee métodos por defecto para las operaciones básicas con componentes (por ejemplo insertar, eliminar, copiar, pegar). En caso de que el diagrama tenga algún comportamiento particular se deberá sobrescribir el método correspondiente.

Clase para datos básicos

Se tiene una clase para los datos básicos de los componentes. Con datos básicos nos referimos a aquellos datos que no son datos gráficos. Por ejemplo, son datos básicos el nombre y el estado de pendiente o no. Todos los datos básicos de un componente son compartidos por todas las instancias de un componente. Es decir, cada componente va a tener cierta información gráfica particular para cada diagrama donde aparezca éste y la información básica que será única.

Por lo tanto, todo componente tendrá un objeto de clase DatosBase o de una subclase de ésta. En caso de que el componente tenga algún otro dato, se deberá extender esta clase para agregar el nuevo campo. Por ejemplo, para el modelo multidimensional, se tiene que los niveles poseen un tipo y por lo tanto, se deberá extender DatosBase para que tenga un campo tipo.

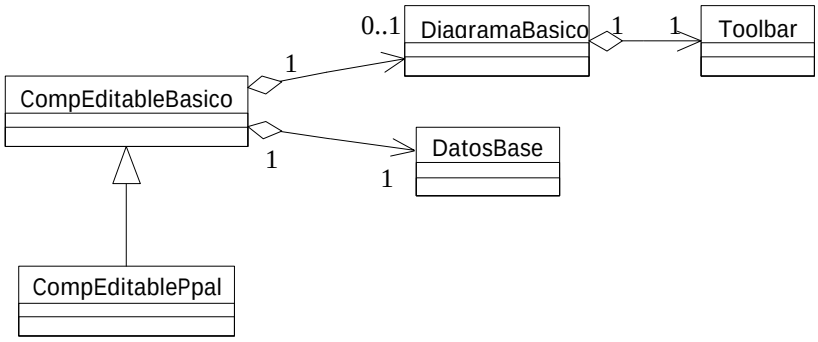

Fig. 13. Diagrama de clases de los componentes editables básicos

3.3. Java

Como ya se había mencionado, dado que el repositorio estaba programado en Java y que era deseable que el editor fuese multiplataforma, era muy factible que el lenguaje de programación para el prototipo fuese Java.

Luego de diseñar el editor, se pensó que seguía siendo la mejor opción. La orientación a objetos de Java permitiría traducir el diseño de las clases de objetos en clases java, además Java proveía los objetos gráficos que se necesitaban para las interfaces gráficas, componentes reutilizables (Java beans) y mecanismos de introspección.

3.3.1. Java beans

Al pensar en extensibilidad y generalidad, surgió inmediatamente la interrogante de si usar o no Java Beans. Luego de discutir y estudiar el tema, el uso de beans parecía ser necesario y no iba a implicar más trabajo.

La arquitectura JavaBeans, define un conjunto de reglas y los java beans son simplemente objetos java que siguen estas reglas. Los java beans, conformes a la API JavaBeans y a patrones de diseño pueden ser reconocidos y manipulados en ambientes de desarrollo de aplicaciones visuales. Esta estandarización, permite que las herramientas reconozcan las características de un bean y usarlo sin saber cómo está implementado.

Además, las clases de Java para reflexión, que permiten inspeccionar objetos y manipularlos en tiempo de ejecución, permiten que una herramienta de desarrollo pueda analizar el bean, cambiar los valores de sus propiedades e invocar sus métodos.

Esto es lo que se necesitaba para tener componentes editables genéricos. La herramienta que analizaría los beans en tiempo de ejecución sería este editor. Luego, no importaría como estuviesen implementados los nuevos componentes a editar, funcionaría si estos cumplen con ciertos requisitos y proveen ciertos servicios.

Por lo tanto, cada componente editable será un bean y éste deberá seguir las reglas de JavaBeans y aquellas particulares del editor.

3.3.2. Beanbox

Una beanbox es una ventana a la cual podemos agregar beans y manipularlos.

Los objetos que están compuestos por otros, tendrán un diagrama que será una beanbox. Por ejemplo, para editar una dimensión, se tendrá una beanbox donde estarán los niveles, las jerarquías y los links que la componen.

El BDK (Beans Development Kit) trae una beanbox para poder probar los beans que se programen. Este programa permite agregar nuevos beans, estos aparecen en una toolbar, luego pueden ser agregados a una ventana y manipulados (cambiados de tamaño, si lo permiten, movidos de lugar, etc), también se muestran las propiedades del bean que está seleccionado, pudiéndole cambiar sus valores y se puede ver los métodos y eventos que manejan, incluso hacer pruebas del comportamiento ante eventos disparados por otros. Se pensó en tomar ideas de la beanbox que viene con BDK 1.1. e incluso en reutilizar código fuente. Para esto, se realizó un estudio de cómo funciona y cómo está implementada (Anexo VI: Estudio de Beanbox). Esto, si bien dio una buena base para la implementación, (tomando ideas de cómo resolver algunos temas: cómo seleccionar beans, cómo insertarlos) es la razón de haber tomado algunas decisiones que más tarde se mencionarán (en sección Implementación del prototipo, subsección Wrapper)

4. Implementación del prototipo

4.1. Ambiente de desarrollo

Para la implementación del prototipo se utilizó Java como lenguaje de programación (jdk1.2.2, java development kit 1.2.2) con la herramienta de desarrollo JBuilder 2.0 de Borland y Visual J++ de Microsoft.

En un primer momento, se probaron otras herramientas de desarrollo como FreeBuilder, OpenBuilder, Visual Café, pero se optó por Jbuilder por ser la que proveía más comodidades para la programación y permitía configurar la versión de java a utilizar (se quería usar jdk1.2.2). No se pudo usar el diseño de formularios de forma gráfica, ya que sólo funciona bien si usa las clases de Borland de formularios y layouts.

En una etapa más avanzada de la programación, se probó utilizar VisualJ++. Se pudo configurar VisualJ ++ para usar jdk1.2.2 y se probó el código generado en plataformas unix (por temor a que no fuese código java puro). Además, resultó ser más amigable, proveía más ventajas para programar y era más rápido (para compilar, para ver los componentes de una clase o un paquete, etc.). Se tuvo el mismo problema con el diseño de los formularios ya que VisualJ ++ usa para esto clases de Microsoft, pero de todas formas, se optó por cambiar a ésta herramienta.

4.2. Prototipo

El prototipo sigue las especificaciones del diseño, correspondiéndose los objetos con las clases java.

Permite ser configurado para editar cualquier modelo que haya sido implementado para él (que siga las especificaciones necesarias para ser un “modelo editable” por este editor).

Las funcionalidades para facilidades de uso fueron cubiertas básicamente permitiendo una cómoda edición del modelo pero quedando pendientes algunas utilidades como por ejemplo el seleccionar varios componentes para copiar o mover.

Se instanció el editor para el modelo de esquemas multidimensionales programando las clases necesarias para esto. Algunos objetos del CMDM como las restricciones o los tipos de los niveles, fueron simplificadas quedando su desarrollo pendiente para una siguiente versión. Por ejemplo, las restricciones son un string (con cualquier tipo de símbolos) que representa la especificación de la restricción en un cierto lenguaje de restricciones pero no se implementaron las restricciones estándar cuya utilidad es facilitar la definición de éstas. En el caso de los tipos de los niveles, también son actualmente un string que representa al nombre del tipo. En una siguiente versión, se deberá desarrollar una clase específica para los tipos y un editor de éstos para la hoja de propiedades.

Se implementaron algunas herramientas específicas del modelo: el mostrar al esquema multidimensional por las relaciones dimensionales y generar código para conexión con el repositorio.

Para la conexión con el repositorio, se realizó un módulo que genera a partir de un esquema multidimensional, un archivo de texto con un formato definido que se detallará más adelante en la sección Instancia para CMDM.

4.3. Paquetes

En la implementación del prototipo hay dos paquetes principales: Editor e InstanciaModelo. En el primero se encuentran las clases del editor genérico y en el segundo, las clases que corresponden a la instancia del modelo de esquemas multidimensionales.

Dentro del paquete Editor hay paquetes con clases para algunas utilidades (como para el cálculo de posiciones de links, etc.). También aparecen otros paquetes con clases para otras utilidades como GeneradorCodigo que tiene las clases correspondientes a la generación de código para la vinculación con el repositorio.

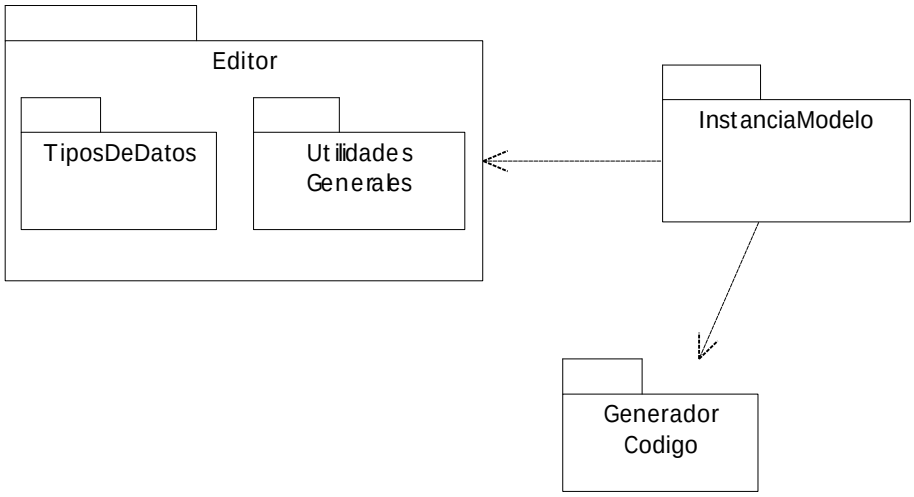

Fig. 14. Paquetes del prototipo del editor.

4.4. Notas sobre la implementación

Todos los componentes editables son beans y por lo tanto deben seguir los estándares de JavaBeans además de aquellos que requiere el editor. Además, todo componente editable debe proveer su clase *BeanInfo* para dar el ícono que se deberá usar en las toolbars y en el nodo del visor de componentes, así como también para indicar las propiedades que permitirá editar (Seguramente no se desee mostrar y permitir editar todas las propiedades del bean).

4.4.1. Wrapper

Como se reutilizó código de la beanbox del bdk1.1, se introdujo la clase Wrapper. Esta es una clase cuyo cometido es uniformizar algunos comportamientos de los beans. Dado que la beanbox de bdk1.1 permite editar cualquier bean, entonces crea por cada bean un objeto wrapper que lo "envuelve", agregándole métodos e interceptando eventos, lo que requiere la beanbox para manipular el bean. (Esto está más detallado en el Apéndice VI: Estudio de la Beanbox).

Dado que en el editor, todos los beans son subclase de un bean básico editable, los métodos o servicios que el editor requiere podrían haberse puesto todos en el bean básico y prescindir del wrapper. Esto podría ser una modificación para una nueva versión del editor. La razón por la cual esto es mencionado como una posible modificación es no sólo porque sea inútil la existencia de esta clase, sino porque de cierta forma, restringe la existencia de algunos tipos de beans.

Dicha restricción surge a partir de lo siguiente: el Wrapper es subclase de Panel y por lo tanto su forma es rectangular. Los beans son agregados a un Wrapper y luego éste Wrapper es insertado en la beanbox. Entonces, lo que hacemos en la beanbox (o diagrama), es manipular al Wrapper y no al bean directamente, y estamos viendo al Wrapper con el bean dentro. Por ejemplo, al seleccionar el bean, lo que se hace es modificar el color de fondo del wrapper para mostrar que está seleccionado y al hacer click con el mouse sobre el wrapper es que se selecciona el bean. El mouse click es interceptado por el wrapper y luego lo atiende el bean. Esto, en el caso de un link, es un problema ya que la superficie del link en la beanbox sería toda la ocupada por el wrapper (que es rectangular) y no sólo por la línea del link. Esto hizo que tuviésemos un tratamiento especial para los links, no insertándolos en un wrapper.

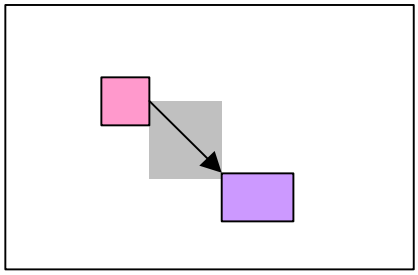

Fig. 15. Cómo se vería un link seleccionado dentro de un Wrapper en un diagrama.

4.4.2. Links

Cómo los links debieron tener un tratamiento distinto al de los demás componentes de un modelo (por lo mencionado en la sección anterior: Wrapper), se implementó una clase básica para éstos objetos. Todo link del modelo será subclase de *BeanLink*. Esta clase, es subclase de BeanCompBasico (como todos los componentes de un modelo), pero además tiene datos importantes que tienen que ver con el origen y destino del link.

Estos datos específicos para los links, son fundamentales para poder ser dibujados y manipulados dentro del diagrama y el hecho de que todo link deba ser subclase de BeanLink, uniformiza el tratamiento de estos objetos en las beanboxs. (Ver Anexo: Manual de instanciación)

5. Instancia para CMDM

Como se decidió que la instanciación del editor para un modelo sería extendiendo las clases básicas, se debió realizar esto para el modelo de esquemas multidimensionales.

5.1. Clases implementadas

5.1.1. Componentes

A partir del análisis del modelo CMDM surgieron las siguientes clases:

Datos base con restricciones: como todos los componentes del editor tienen asociadas restricciones, entonces todos los objetos tendrán la propiedad restricción. Es así que se extendió la clase de datos base agregándole las restricciones.

Esquema: subclase del componente editable principal. En esta clase se indicarán los componentes que tendrá el esquema multidimensional (nivel, dimensión, relación dimensional, jerarquía) y las herramientas y se darán los métodos para cada una de las herramientas del modelo. No se dará beanbox para este objeto y como datos base usará la clase de datos base con restricciones.

Nivel: subclase de componente editable básico. Esta clase tampoco tendrá un diagrama asociado pero tendrá una nueva clase para datos base porque tiene una nueva propiedad: tipo. Esta clase de datos base será extensión de datos base con restricciones.

Jerarquía: subclase de componente editable básico. Esta clase tendrá un diagrama asociado BeanboxJerarquia. La clase de datos base que usará será la de datos base con restricciones.

BeanboxJerarquia: extensión de BeanboxBasica. En la toolbar se agregarán los componentes: jerarquía macro, nivel y link.

Dimensión: subclase de jerarquía. Esta clase tendrá un diagrama asociado BeanboxDimension. La clase de datos base que usará será la de datos base con restricciones.

Jerarquía macro: subclase de jerarquía. Esta clase tendrá un diagrama asociado BeanboxJerarquíaMacro. La clase de datos base que usará será la de datos base con restricciones.

Relación dimensional: subclase de componente editable básico. Esta clase tendrá un diagrama asociado BeanboxRelDim. La clase de datos base que usará será la de datos base con restricciones.

LinkCompJerarq: subclase de BeanLinkFlecha.

LinkDims: Subclase de la clase BeanLink.

LinkDimMedida: Subclase de BeanLinkFlecha.

```
public class BeanJerarquia extends BeanCompBasico
                                    implements Serializable{
  public BeanJerarquia() {
        setColor(Color.magenta);
        setDatosBase(new DatosBaseConRestricciones());
       setDiagrama(new BeanboxJerarquia(this, getNombre()));
  }
}
```

Fig. 16. Ejemplo de instanciación para el bean jerarquía del modelo multidimensional

5.1.2. Diagramas

Para los componentes con diagramas, por cada uno se extendió la beanbox básica, indicando en esta nueva clase, por ejemplo, los objetos que irán en su toolbar, y debiendo sobrescribir algunos métodos para algunos diagramas con comportamientos más particulares (como en el caso de las relaciones dimensionales).

Para no extender esta sección demasiado, sólo se mencionará el caso del diagrama de una jerarquía que es muy simple y el de la relación dimensional, el cual tiene un comportamiento distinto al resto de los diagramas. (Para más detalles sobre la instanciación ver Anexo VII: Manual de instanciación.)

BeanboxJerarquía:

Es una extensión de la beanbox básica y se deberán agregar los componentes de la toolbar que son nivel, jerarquía y link. Esto es mediante el método de la BeanboxBasica: *agregarAToolBar* que recibe un objeto de clase Button, el nombre completo de la clase del componente y una referencia al objeto que maneja el evento de presionar el botón. Luego se sobrescribe el método actionPerformed para atender los eventos de la toolbar que en este caso lo único que deberá hacer es invocar al método de BeanboxBasica *agregarBean* o en el caso de un link el *agregarBeanLink.*

BeanboxRelDim:

Para el caso del diagrama de las relaciones dimensionales, se tuvo que tener en cuenta la representación particular de una relación dimensional que es con una elipse central y luego todas las dimensiones que la componen unidas a ésta por un link el cual tendrá o no una punta de flecha según sea o no la dimensión para la medida del cubo asociado.

En este caso se tuvo que extender el panel de la beanbox para sobrescribir un método auxiliar del método de atención al evento mouse_released para crear el link de la dimensión a la elipse cada vez que se inserta una dimensión.

```
public class BeanboxJerarquia extends BeanboxBasica {

  JButton butNivel = new JButton();
  JButton butJerarquia = new JButton();
  JButton butLink = new JButton();

  public BeanboxJerarquia(BeanCompBasico padre,String nombre) {
     super(padre, "Jerarquia - " + nombre);
     setTitle("Jerarquia - " + nombre);

     agregarAToolBar(butNivel, "Editor.BeanNivel", this);
     agregarAToolBar(butJerarquia, "Editor.BeanJerarquiaMacro", this);
     agregarAToolBar(butLink, "Editor.BeanLinkFlecha", this);
  }

  public void actionPerformed(ActionEvent evt) {

    if (evt.getSource()== butNivel) {
      agregarBean("Editor.BeanNivel");
      return;
    }
    if (evt.getSource()== butJerarquia) {
      agregarBean("Editor.BeanJerarquiaMacro");
      return;
    }
    if (evt.getSource()== butLink) {
      String nombreNuevo = "Link" + cantLinks;
      BeanCompBasico comp = instanciarBean(getPadre(),
                                  "Editor.BeanLinkFlecha", nombreNuevo);
      agregarBeanLink(comp, nombreNuevo, "Editor.BeanLinkFlecha");
      cantLinks ++;
      return;
    }
  }
  private int cantLinks = 0;
}
```

Fig. 17. Ejemplo de insanciación para la beanbox de jerarquía.

5.2. Cómo se guarda un esquema

La forma en que se guarda el objeto editado es responsabilidad del programador del modelo. Al pedir guardar los datos, lo que el editor hace es llamar al método guardar del componente principal.

Cuando se pensó en cómo lograr la persistencia de los objetos editados, primero se consideró hacerlo con el mecanismo que provee java de serialización. La serialización de java es un mecanismo que permite generar un archivo con la información de un objeto (el objeto serializado), sea cual sea la complejidad de este objeto y luego a partir del archivo, regenerar el objeto. Java provee los métodos por defecto para serializar y deserializar un objeto, pero sólo sirve para objetos simples, debiendo ser sobrescrito para objetos complejos.

El objeto a guardar en este caso era complejo, entonces debía ser sobrescrito el método de serialización y no era simple la manera en que debía hacerse ya que cada objeto, además de tener propiedades, tenía otros objetos que lo componían con propiedades locales (color, tamaño, etc.) y un diagrama donde estaban dispuestos éstos.

Además, debía haber alguna forma de interfase con el repositorio, y se necesitaba tener algún archivo de formato conocido con todos los datos del esquema. Por lo tanto, se decidió guardar el objeto editado en un archivo de texto. Es así que, guardar un esquema o generar el código para el repositorio coinciden (por el momento, ya que como los métodos invocados son distintos, aunque hagan lo mismo, estos podrían cambiar en distinta forma).

El archivo de texto que se genera al guardar un esquema, tiene el formato que se describirá a continuación.

La idea principal es que cada componente y grupo de componentes de un determinado tipo, estará en un bloque delimitado por las palabras reservadas “begin” y “end”. El bloque mas externo, corresponde al bloque del esquema, y tiene la siguiente forma:

begin <Nombre_esquema>
< Propiedades globales>
<Restricciones>
begin <Nombre de grupo de componentes 1>
begin <Componente 1>
<Propiedades globales del Componente 1>
<Restricciones del Componente 1>
begin <Nombre de grupo de componentes 1>
begin <Componente 1>
<Propiedades locales del componente>
end
begin <Componente 2>
<Propiedades locales del componente>
end
...
end
begin <Nombre de grupo de componentes 2>
begin <Componente 1>
<Propiedades locales del componente>
end
begin <Componente 2>

<Propiedades locales del componente>
end
…
end
...
end
...
end
begin <Nombre de grupo de componentes 2>
... (Análogo)
end
...
end

En el lugar indicado por “propiedades globales”, se encuentra una lista de propiedades globales que tiene la siguiente forma:

GlobalProperty <Nombre propiedad> <Lista de valores> **end**

En el lugar indicado por “restricciones”, se encuentra el código correspondiente a las restricciones del componente y el formato es:

Restriction <String de especificación de la restriccion> **end**

Las propiedades locales, tienen un formato similar a las globales:

LocalProperty <Nombre propiedad> <Lista de valores> **end**

La lista de palabras reservadas es:

begin, end, GlobalProperty, LocalProperty, Restriction.

Es importante que en los valores de las propiedades no aparezcan ninguna de estas palabras, pues se generarían conflictos.

Con respecto a los símbolos permitidos:

En las restricciones se puede utilizar cualquier tipo de símbolos. La única excepción es la palabra reservada end.

En el resto del código, los símbolos permitidos son:

A..Z, a..z, 0..9, “Ñ”, “ñ”, “á”, “é”, “í”,“ó”, “ú”, “_”, “.”, “(”, ”)”, ”&”, ”$”, “-”, “+”.

```
begin Ventas
    GlobalProperty pendiente false end
    Restriction Restricciones del esquema ventas end
    begin Niveles
        begin Dia
            GlobalProperty pendiente false end
            GlobalProperty tipo  end
            Restriction (Restr de día)  end
        end
        begin Mes
            GlobalProperty pendiente false end
            GlobalProperty tipo  end
            Restriction  end
        end
        begin Cuidad
            GlobalProperty pendiente false end
            GlobalProperty tipo  end
            Restriction  end
        End
        ……
    begin Dimensiones
          begin Ventas
               GlobalProperty pendiente false end
               Restriction  end
               begin Editor.BeanNivel
                    begin Sucursal
                         LocalProperty locationWrapper 121 42 end
                         LocalProperty size 61.0 24.0 end
                          LocalProperty color -256 end
                     end
                     begin Vendedor
          ……..
```

Fig. 18. Ejemplo: parte del texto generado para un esquema multidimensional

Descripción formal del código generado:

<Esquema> → begin <Nombre> <Cuerpo_esquema> end

<Cuerpo_esquema> → <L_Props_Globales> <Restricciones>
<L_Grupos_Componentes>

<L_Props_Globales> → GlobalProperty <Nombre> <L_Valores> end
<L_Props_Globales>
| ε

<Restricciones> → Restriction <String> end

<L_Grupos_Componentes> → begin <Nombre> <L_Componentes> end
<L_Grupos_Componentes>
| begin <Nombre> <L_Componentes> end

<L_Componentes> → begin <Nombre> <Cuerpo_Componente> end

<L_Componentes>

| begin <Nombre> <Cuerpo_Componente> end

<Cuerpo_Componente> → <L_Props_Globales> <Restricciones>
<L_Grup_Comp_Integrantes>

<L_Grup_Comp_Integrantes> → begin <Nombre> <L_Comp_Int> end
<L_Grup_Comp_Integrantes >

| ε

<L_Comp_Int> → begin <Cuerpo_Comp_Integrantes> end < L_Comp_Int>

| begin <Cuerpo_Comp_Integrantes> end

<Cuerpo_Comp_Integrantes> → <L_Props_Locales>

| <L_Props_Locales> <Restricciones>

<L_Props_Locales> → LocalProperty <Nombre> <L_Valores> end
<L_Props_ Locales >

| ε

<L_Valores> → <Palabra> <L_Valores> | <Palabra>

<Nombre> → <Palabra>

<Palabra> → <Car_Permitido> <Palabra>

| <Car_Permitido>

<Car_Permitido> → a | ... | z

| A | ... | Z

| 0 | .. | 9

| Ñ | ñ | á | é | í |ó | ú | _ | . | (|) | & | $ | - | +

<String> → <Car> <String> | <Car>

<Car> → (Cualquier caracter ASCII)

5.3. Herramientas particulares del modelo

Se implementaron dos herramientas propias del modelo que son: generar código para el repositorio y ver el esquema multidimensional por las relaciones dimensionales.

Para esto, se programaron los métodos para ejecutar estas herramientas y luego en la clase del componente básico que es el BeanEsquema, se indicó qué herramientas provee. Esto se hace mediante el método del objeto BeanCompPrincipal *agregarHerramienta* que recibe como parámetros la descripción que se quiere que aparezca en el item del menú correspondiente a esta herramienta y el nombre del método del componente principal que la ejecuta. Ejemplo:

```
agregarHerramienta(" Generar código para repositorio",
                                                "genCodigoRepositorio");
```

6. Extensibilidad

El editor está diseñado para poder editar cualquier modelo dentro de ciertas características. Para editar un nuevo modelo, se deberá programar las clases que representen los componentes de éste. El editor provee objetos, métodos y comportamientos por defecto, requiriendo muy poca programación para un modelo que se ajuste a estos comportamientos, y necesitando mayor programación a medida que el modelo se aleje más de éstos.

Por lo tanto, el editor instanciado para un modelo, tiene mucha flexibilidad para ser extendido. Analicemos algunos posibles casos en que se quisiera extender el editor:

- *Agregar un nuevo componente:* es simple agregar un nuevo componente, de hecho es como cuando se agrega el primer componente. Dependerá de cuanto se ajuste al comportamiento por defecto, el trabajo de programación que requiera. Podría llegar a ser simplemente extender la clase de componente editable básico y agregarla a la lista de componentes del componente principal del modelo.
- *Agregar una nueva herramienta:* para agregar una nueva herramienta habría que programar el método que la ejecute en el componente principal y agregarla a la lista de herramientas.
- *Agregar una propiedad gráfica a algún objeto*: agregarla a la clase del objeto (con el estándar de get y set que exige Java Beans) y luego a la clase de BeanInfo del objeto (para poder editarla en la hoja de propiedades)
- *Agregar una propiedad no gráfica (compartida por todas las instancias) a un objeto:* primero agregarla a la clase del objeto (con el estándar de Java Beans), luego, agregar la propiedad a la clase datos base que use (con el estándar de Java Beans). Para esto hay que tener en cuenta dos casos: 1) si tiene una clase de datos base particular para ella, simplemente se agrega la propiedad, 2) sino, extender la clase de datos base que usa y agregar la propiedad a esta clase y luego indicarle a la clase del objeto cual es la clase de datos base que usa ahora.
- *Agregar un nuevo diagrama:* Para agregar un nuevo diagrama, se debe extender la beanbox básica. Luego, en el método de creación de la clase nueva, agregar los beans que tendrá en su toolbar (ya sean otros componentes o herramientas) y luego programar el cuerpo de la atención de los botones de la toolbar. Esta rutina de atención a los botones, en caso de ser un bean que va a ser agregado a la beanbox, simplemente será una llamada a una función de insertar bean por defecto, si no, deberá programarse lo que deba hacer.

7. Conclusiones. Extensiones futuras.

7.1. Conclusiones

Se construyó no sólo un editor gráfico para editar esquemas multidimensionales, sino un editor genérico.

El prototipo permite editar cualquier modelo que esté implementado, simplemente indicando cual es el objeto principal.

En este proyecto, se instanció el editor con el modelo de esquemas multidimensionales CMDM. Se pueden crear esquemas y conservarlos exportándolos a un archivo de texto (con un formato bien determinado) que será la forma de comunicación con el repositorio. También se pueden leer a partir del archivo, pudiendo ser consultados y modificados. También se implementaron algunas herramientas específicas del modelo, lo que sirve como ejemplo para mostrar cómo agregar una nueva herramienta.

Las funcionalidades para facilidades de uso fueron cubiertas básicamente quedando algunas utilidades pendientes (como por ejemplo seleccionar y copiar varios objetos).

Algunos objetos del CMDM, fueron simplificados, quedando su desarrollo pendiente para una siguiente versión. Por ejemplo, las restricciones son actualmente un string que representa su especificación. Su editor para la hoja de propiedades es particular pero muy simple (un cuadro de texto con barras de desplazamiento). En una siguiente versión pasaría a tener un editor más complejo incorporando las restricciones estándar y utilidades de arrastrar y pegar para simplificar su definición.

7.2. Extensiones futuras

7.2.1. Extensiones a nivel del editor

Consideramos las siguientes utilidades como extensiones futuras para el editor :

- Impresión de los diagramas y de los objetos (sus propiedades).
- Implementar los utilitarios de deshacer y rehacer. Para ello se podría tener, para cada operación dentro de la edición del esquema, una operación “contraria” que deshiciera a ésta e ir guardando una determinada cantidad de operaciones contrarias correspondientes a las últimas operaciones realizadas.
- Poder hacer selección múltiple para mover, cortar, copiar y pegar.
- Hacer “wizards” para programar un nuevo modelo, para no tener que programar directamente en java.
- Implementar una utilidad que informe de los componentes que están pendientes. Poder configurar si por defecto un componente se crea pendiente o no.

7.2.2. Extensiones a nivel del modelo

A nivel de la instancia para CMDM, los siguientes puntos deberían ser considerados para una siguiente versión:

- Implementar un editor para restricciones para tener restricciones estándar y con utilidades de arrastrar y pegar. Para ello se deberá realizar un editor de restricciones más complejo, con toolbars, con componentes gráficos que representen a las restricciones estándar (macros) que puedan incorporarse a la especificación de la restricción con sólo arrastrar.
- Agregar restricciones sobre qué componente se puede agregar en qué diagrama y qué relaciones puede haber entre ellos. Para ello, se deberán incorporar funciones de chequeo previas a las utilidades de pegar e insertar.
- Desarrollar el objeto jerarquía con las utilidades que se vieron en el análisis:
    - Implementar jerarquías con parámetros.
    - Selección y operaciones dentro de una jerarquía:
        - Dado un nivel, referenciar a todos los niveles hacia arriba.
        - Dados 2 niveles referenciar a la jerarquía entre ellos (los niveles deberán estar en la relación jerárquica)
        - Una vez seleccionada una subjerarquía, poder definirla como una jerarquía independiente para reutilizar (una nueva jerarquía macro).
- Compilación de restricciones (para hacer los chequeos sintácticos de sus especificaciones). Se podría crear una herramienta que realizara el chequeo de todas las restricciones del esquema o de las que se seleccionasen, o se podría pedir el chequeo sintáctico de la restricción dentro del propio editor de restricciones.

8. Bibliografía

[UML1] G. Booch - I. Jacobson - J. Rumbaugh. The Unified Modeling Language. User Guide .

[UML2] Pierre Alain Muller. Instant UML.

[CMDM] F. Carpani. CMDM: Un modelo conceptual para Data Warehouse.

[Core] Kim Topley Core JFC.

[Sw99] Sun Swing Tutorial Ago. 1999.

[JBas] Monica Pawlan. Java Programming Language Basics.

[JBShC] MageLang Institute. Java Beans Short Course. Abr. 1999.

[BTut2] Java Beans Tutorial. Nov. 1997.

[BTut] Beans Tutorial. Mar. 1997.

[ExpJ] Patrick Niemeyer & Joshua Peck. Exploring Java. Set. 1997.

[JNut] David Flanagan. Java in a Nutshell. May. 1997.

9. Anexos

Anexo I: Análisis inicial.

Documento de análisis inicial en el que se realizó el análisis del modelo CMDM.

Anexo II: Análisis detallado.

Documento de análisis detallado que incluye el análisis de requerimientos del editor.

Anexo III: Diseño general.

Anexo IV: Diseño detallado.

Anexo V: Estudio de Java y Java Beans.

Breve resumen con algunos temas del estudio de java y java beans.

Anexo VI: Estudio de la beanbox de BDK 1.1.

Documento realizado luego del estudio de la beanbox de BDK 1.1 (beans development kit)

Anexo VII: Manual de instanciación.

Documento con detalles de la implementación del editor y explicaciones de cómo instanciar el editor para un nuevo modelo, con ejemplos de la instancia para esquemas multidimensionales.

Nota: Los anexos I al VI son documentos hechos durante el transcurso del proyecto.