SOFTWARE DE AYUDA AL DISEÑO CON CAPACIDAD DE VERSIONAMIENTO DE OBJETOS

Lic. Ana María García Pérez *
Dra. Luisa González González*
Ing. Ionel Muñoz Jiménez **
Lic. Niriam Peña Valdés***
Lic. Lidia R. Ríos ***

RESUMEN

Se presenta un modelo de versiones que sirve para la construcción de software de ayuda al diseño.

Este software comúnmente automatiza la obtención de una variante, dado un proyecto. Por ejemplo, permite crear un plano para un circuito, un plano para un edificio, etc. Si se desean crear varias variantes, el diseñador debe almacenarlas en archivos distintos y no existe manera de operar sobre las versiones desde el propio software, como no sea compararlas a la vista abriendo los archivos correspondientes.

Todo el control de versiones debe hacerse manualmente y de manera independiente, digamos, para saber qué versión se derivó de cuál o cuál fue la que se obtuvo con costo menor.

Además, cada versión del mismo objeto de diseño almacena toda la información, aunque existan valores iguales entre ellas, puesto que se identifica una versión con un archivo creado por el sistema.

El modelo propuesto permite:

- Crear ilimitadas versiones de un proyecto (acotadas sólo por las capacidades de almacenamiento) minimizando la información física que se repite y sin necesidad de que el usuario grabe cada versión en un archivo.
- Recuperar versiones por medio de solicitudes declarativas.
- Controlar la historia de las derivaciones.
- Almacenar versiones con diferencias en la estructura para un mismo objeto de diseño.

INTRODUCCIÓN

En los procesos de diseño se reconoce como cualidad deseable el poder decidir el prototipo o versión definitiva luego del estudio de varias variantes. Si un software automatiza un proceso de optimización basado en cierto criterio (p. ej. la versión con costo mínimo) otros criterios (como rendimiento, requisitos de velocidad, etc.) se

* Departamento de Computación Universidad Central de Las Villas
** ESI Camagüey
*** ESI Sancti Spíritus

quedan ocultos y el diseñador no tiene el control total sobre el proceso. De ahí que la elaboración de software que permita que el diseñador tenga "el control total" basado en la evaluación de un conjunto de versiones, es un propósito deseable y en computación introduce un modo novedoso de trabajo, pues no es lo tradicional.

Se pretende entonces que se identifique el diseño de un objeto como un proyecto, el cual sí tendrá nombre de archivo; las versiones de este objeto se pueden crear a partir de *revisiones* de una versión inicial que crea el propio diseñador. Estas versiones se visualizan constantemente con un identificador en el ambiente donde se hace el diseño, de manera que el usuario puede "regresar atrás" para ver versiones previas e incluso puede utilizarlas en la obtención de la próxima versión. Cada operación de salvado no destruye la versión sobre la que se trabaja, sino que introduce una nueva.

Las investigaciones alrededor de la capacidad de versionamiento en el software se han iniciado en las bases de datos temporales, el CASE y el CAD. En la literatura aparecen reportados modelos teóricos (fundamentalmente dirigidos hacia CAD y bases de datos temporales) y en CASE se reportan productos que intentan organizar configuraciones para la obtención de software (GCS- Gestión de Configuraciones de Software) basadas en versiones de módulos de programa.

Lo más positivo de la teoría desarrollada es haber definido un conjunto de conceptos básicos para este problema y haber delimitado claramente el alcance del versionamiento de datos en comparación con la *modificación* y la *evolución*.

Se observa el uso de terminología diferente para denotar conceptos similares, aunque un trabajo importante (el de Katz90) se considera un excelente resumen de modelos de versiones para CAD con el propósito de unificar terminología.

Aunque se reconoce que una versión es "una instantánea significativa" de un objeto de diseño (KATZ90, BER93), en varios modelos, por ejemplo, en el de Sciore (SCIO91) se considera a una versión como instancia de un esquema de objeto, y por lo tanto, todas las versiones de un objeto tienen el mismo esquema. Esto impide que el diseñador haga cambios a la estructura y estos cambios se consideren simplemente como versiones del mismo objeto.

Productos como ORION (KIM89) que se consideran de los más avanzados en versionamiento, no permiten tratar como versiones del mismo objeto a instancias con diferencias en la estructura.

Los productos para CASE aún no se integran al propio ambiente de desarrollo del software, pues son herramientas independientes.

El software de ayuda al diseño conocido en Cuba no maneja en general versiones, es decir, no presenta ninguna de las características que con este modelo se aportan, por lo que el desarrollo de aplicaciones a partir de lo obtenido tiene el valor práctico de permitirle al diseñador mantener versiones para un proyecto y operar con ellas como si fueran datos, en lugar de mantener archivos para un proyecto (con la imposibilidad de acceder a sus datos si no se abren estos archivos). Este modelo también puede utilizarse para añadir control de versiones a otro software que ya exista pero que cree archivos en lugar de versiones, siempre y cuando la estructura de estos archivos sea conocida.

Uso de Sistemas de Gestión de Bases de Datos para aplicaciones de diseño

Los Sistemas de Gestión de Bases de Datos (SGBD) han sido desarrollados para aplicaciones en que grandes volúmenes de datos son tratados y compartidos por usuarios que no desean conocer los detalles de su almacenamiento y organización. Como las aplicaciones CAD tienen estas características, muchos proyectos se han apoyado en SGBD comercialmente disponibles para incrementar la productividad de sus sistemas, encontrando dificultades ya que los mismos han sido desarrollados fundamentalmente para bases de datos orientadas a la gestión y no se brindan facilidades importantes requeridas en aplicaciones ingenieriles. Las diferencias entre bases de datos para aplicaciones CAD y las de gestión son consecuencia de las diferencias fundamentales en la naturaleza de los datos y de las operaciones que se realizan sobre estos datos (KET90) de forma que los requerimientos del manejo de bases de datos de aplicaciones ingenieriles son inherentemente diferentes a los de las aplicaciones orientadas a la gestión.

Los SGBD se han utilizado como parte de software para diseño con el fin de apoyar el uso de normas. Para estas resultan adecuados los esquemas relacionales, pero no resulta así para poder representar objetos de diseño, pues los objetos necesitan ser descompuestos en tablas y para reconstruírlos se precisa la ejecución de acoples mediante llaves extranjeras (WIL84).

Por ello, las aplicaciones en bases de datos se han concentrado en el desarrollo de mecanismos que permitan describir, de la manera más natural y directa posible, las propiedades estructurales de una aplicación, sobre todo cuando éstas han de manejar objetos de datos complejos.

El concepto de *Objeto Complejo* ha sido introducido en algunos OO-DBMS (del inglés: Sistemas de Gestión de Bases de Datos Orientados a Objeto), por ejemplo ORION (KIM89) y en algunos lenguajes de programación (SHA89) para permitir a las aplicaciones modelar el hecho de que varios objetos (conocidos como objetos componentes) constituyan una entidad lógica.

La idea de un *SGBD orientado a objetos* (SGBDOO) es almacenar los objetos como tales, y así salvar el puente semántico entre el modelo de objetos y la base de datos como tal.

En esta forma, los programas no tienen que ejecutar los acoples lentos que son necesarios para recuperar a un objeto completo en un SGBD relacional. Esto significa que los SGBDOO son muy útiles en las aplicaciones que requieren manejar objetos complejos persistentes.

Sin embargo, a pesar de las indudables ventajas de los SGBDOO para la manipulación de objetos complejos persistentes, aún es débil la capacidad de versionamiento de los objetos que se ha logrado incluír en las aplicaciones, pues no se reportan prácticamente.

Modelación de versiones.

Para que un sistema controle versiones, se requiere que maneje tanto desde el punto de vista lógico como físico, un conjunto de conceptos que deben ser organizados adecuadamente. Primeramente, pasemos a estudiar el alcance del versionamiento. Para

ello es preciso evaluar los diferentes estadíos que un sistema de bases de datos puede alcanzar en cuanto a la manipulación de los esquemas y de los datos.

Existen tres estadíos en lo que a esquemas se refiere (ROD94):

Modificación de esquemas: Cuando el SGBD permite cambios a la definición del esquema de la base de datos.

Evolución de esquemas: Cuando el SGBD permite la modificación del esquema sin perder el contenido semántico de los datos existentes.

Versionamiento de esquemas: Cuando el SGBD permite la visión de todos los datos, tanto prospectiva como retrospectivamente, a través de interfaces definibles por el usuario para las versiones.

La evolución de esquemas no necesariamente involucra soporte histórico de los cambios, sin embargo, versionamiento, aún en su forma más simple, requiere que se mantenga una historia de los cambios para permitir la retención de la definición de esquemas pasados.

El versionamiento incluye que no se pierda información valiosa si por ejemplo, cambia el dominio de un atributo. Para ello es preciso definir reglas de integridad y mecanismos que controlen la evolución.

Conceptos básicos de un problema de versiones

En bases de datos para aplicaciones de diseño, un *objeto de diseño* es una agregación de datos que es tratada como una unidad coherente por los diseñadores. Un objeto de diseño es usualmente un objeto complejo, el cual se obtiene ensamblando objetos que pueden ser o bien primitivos u objetos complejos. Por ejemplo, en ingeniería de software, un objeto de diseño puede ser un paquete de software que consiste de módulos de código fuente, librerías y manuales de documentación. En diseño VLSI, un objeto de diseño puede ser una ALU (unidad aritmética y lógica) que consiste de una agregación de máscaras geométricas, circuitos lógicos y programas escritos en código de máquina. En diseño mecánico, el objeto de diseño puede ser una pieza compuesta de partes, como un troquel, mecanismos de trasmisión, una bicicleta, etc. En hipertextos, el objeto de diseño es un documento.

Una *versión* es una instantánea significativa de un objeto de diseño creado en un momento del tiempo. Obviamente, el concepto de «semánticamente significativa» no puede ser definido ambiguamente para todas las aplicaciones.

A pesar de que una versión de objeto de diseño es una instancia de un modelo objeto, es deseable poder conservar el *estado del diseño* en cualquier momento, es decir, sin estar aún terminado, por lo que una versión del objeto puede también consistir de un esquema del objeto junto a los valores de las propiedades de componentes que se hayan alcanzado a tener en ese momento.

Aparte de la semántica atribuída al concepto de versión, una versión de un objeto dado se crea a partir de las modificaciones hechas en el tiempo a las versiones previas, partiendo de una versión inicial. Un esquema de estas derivaciones puede apreciarse en la *historia de versiones,* cuya representación gráfica más natural es un árbol (ver fig. 1). Los nodos del árbol identifican las versiones

y los arcos representan las *interrelaciones de derivación*.

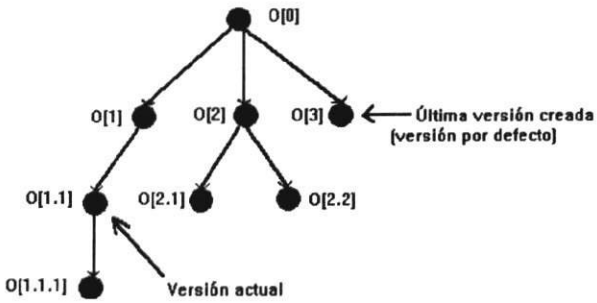

Fig. 1 - Historia de una versión (Tomado de BER93)

Varias versiones paralelas pueden ser derivadas a partir de una versión dada. Estas versiones paralelas se definen como *alternativas*. En la figura anterior, O[1], O[2] y O[3] son alternativas de la versión O[0].

En general, una versión de un objeto complejo puede consistir de versiones específicas de sus componentes, con este fin se introduce en los modelos el concepto de *configuración*. Una configuración no es más que el enlace que se establece entre una versión de un objeto compuesto y una versión de sus objetos componentes.

Dentro de la historia, se identifica la *versión actual* que no necesariamente tiene que ser la *última*. El sistema debe proporcionarle al usuario las primitivas que le permitan seleccionar la versión actual (con la que desea trabajar) y si el usuario falla, le mantiene una versión actual por defecto que usualmente es la más reciente.

En aplicaciones de diseño, los objetos y sus versiones se organizan en *espacios de trabajo,* de manera que los objetos en un espacio *privado* se distinguen de los que están en un espacio *público,* que se considera un estadío final de procesamiento. Un espacio de trabajo es un repositorio de objetos, que si es privado, posee mecanismos de protección. Para evitar que las versiones caigan en estados «incompletos» debido a que existen ciertos accesos denegados, muchas veces los espacios de trabajo se hacen semi-públicos o simplemente se ignoran.

Otro aspecto de la modelación de versiones es la inclusión de la aplicación de las versiones no sólo a las instancias sino también a los esquemas. De hecho, debido a la naturaleza de las aplicaciones de diseño, un cambio significativo del objeto de diseño puede involucrar no sólo modificar valores sino también modificar la estructura , según se plantea en la literatura [KATZ90),(BER93)].

Limitaciones teóricas y prácticas en cuanto a manejo de versiones en la actualidad

En la mayoría de los modelos de versiones consultados [(KIM89),(DIT88),(KLA86),(LAN86), (KET87), (BEE88), (VIN88), (SUN88), (KATZ90)] se exponen ideas orientadas hacia el desarrollo de software con capacidad de lograr la persistencia y manejo de versiones de objetos, cuyos esquemas se describen de acuerdo a una sintaxis definida de lenguaje de programación, lo cual representa un modo de trabajo no natural para diseñadores sino para programadores de aplicación.

Es decir, se hacen extensiones de lenguajes de programación para aprovechar la posibilidad de modelar directamente la estructura y el comportamiento de los objetos de la realidad utilizando los objetos y el paso de mensajes del lenguaje de programación. Algunos modelos de versiones, por ejemplo, el de Sciore (SCIO91) consideran que las versiones tienen todas el mismo esquema. Otros modelos no distinguen esquemas e

instancias sino que consideran al objeto de diseño como un todo.

El versionamiento se ha investigado en la *Gestión de Configuraciones de Software,* una rama de la Ingeniería de Software. En este momento se encuentran en el mercado varios productos, denominados bajo el título genérico de «Controladores de Versiones de Software».(MIC95,DEL95,BET95,LEACH95).

La característica común es que son ambientes que permiten ver «desde fuera» las configuraciones de los productos de software en desarrollo, es decir, permiten referenciar documentos, bibliotecas, módulos, etc. de una manera organizada y manejan todos el concepto de «proyecto» que viene a ser el correspondiente al ya definido «objeto de diseño». Estos paquetes permiten poca interoperabilidad con el propio ambiente de desarrollo ya que resultan ser herramientas independientes, es decir, separadas de lo que se diseña.

La orientación a objetos, en tanto proporciona construcciones válidas para una representación más natural de los objetos del mundo real, debe aprovecharse en tal sentido, pero se nota que aún deben investigarse nuevos mecanismos de recuperación de la información cuando ésta se representa con objetos, puesto que el acceso es navegacional (de uno en uno) y no soportan la satisfacción de solicitudes declarativas ni es posible la persistencia de objetos con diferentes esquemas si pertenecen a una clase.

Por otra parte, los SGBD relacionales tienen instrumentados mecanismos de recuperación eficientes y probados que se han vuelto clásicos, pero el modelo relacional como modelo lógico no es capaz de permitir la representación directa de objetos complejos de datos, necesarios en los procesos de diseño actuales.

Modelos para el software

En la literatura sobre análisis de sistemas de software aparecen frecuentemente mezclados diferentes aspectos para abordar la solución computacional a problemas.

En la actualidad ha cobrado auge el Análisis Orientado a Objeto [(BOO94), (COA90)]. Sin embargo, por lo general no se distingue el uso de los objetos para propósitos específicos de funcionalidad, comunicación hombre-máquina o como almacenes de datos, sino que todos los objetos pertenecen a clases que se denotan igual en la documentación del operaciones analista.

Una visión diferente aparece, a nuestro juicio, en la obra de Jacobson (JAC92).

Este autor caracteriza a las aplicaciones en tres dimensiones, que para él son requeridas en el modelo de análisis:

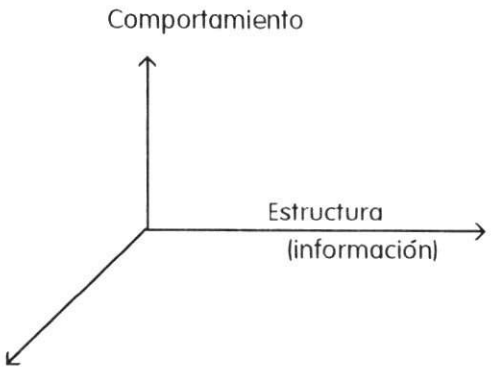

Fig 2 - Las dimensiones de una aplicación

El distinguir claramente estos tres aspectos resulta de vital importancia no sólo para la construcción sino para el mantenimiento de una aplicación.

Cada uno de estos tres aspectos de una aplicación puede representarse por medio de un modelo, de esta manera podemos concebir modelos de interfaz, modelos para la estructura y modelos de comportamiento.

Es objetivo de este trabajo abordar fundamentalmente la modelación de comportamiento, como representación de la evolución de los datos y los metadatos a través de la ejecución en el tiempo de una aplicación. Esto incluye como aspecto a considerar el de la integridad o consistencia pero no como objetivo único o central, sino como algo implícito en el proceso.

Para el diseño de la interfaz de usuario, consideramos como intención principal proveer el control de versiones *en el mismo ambiente de desarrollo*. Varios productos para el control de versiones son sin embargo herramientas con ambiente independiente y esto puede reducir la productividad del proceso de diseño, la generación de ideas y otros procesos mentales.

En la opinión de varios autores [(SHN87), (PRES93), (SOM91)], el principio fundamental que debe tenerse en cuenta para el diseño de interfaces es su *adecuación a las necesidades del usuario*. Este principio se traduce en la división modular de cualquier sistema orientada al problema y el abandono de las tradicionales interfaces orientadas al procesamiento computacional (Entradas, Cálculos, Impresión).

El Modelo Propuesto

La intención principal del modelo es permitir la construcción de un software de aplicación donde el diseñador pueda salvar cualquier estado de diseño que considere importante, y no perder los estados de diseño previos al actual.

Si en un momento dado el usuario decide salvar su trabajo, esto origina la creación de una versión física, que consistirá no sólo de los datos sino también del esquema del objeto de diseño en ese momento.

También se permitirá la recuperación de versiones previas, para lo cual será necesario convertir una versión física en una versión semánticamente comprensible, a lo que se denomina en el modelo "versión lógica", esto implica volver a crear los objetos de programación que permiten visualizar lógicamente la versión seleccionada.

Las transiciones que ocurrirán durante el trabajo se muestran en la figura 3:

1 Conceptualización
2 Recuperación
3 Instanciación
4 Salva

Fig. 3 - La transformación de los niveles conceptual, lógico y físico por medio de los eventos Conceptualización, Instanciación, Salva y Recuperación.

Ello introduce la necesidad de dividir el modelo de versiones en tres niveles: un nivel para definir los conceptos necesarios y las operaciones sobre ellos (nivel conceptual), un nivel para dejar abierto el mecanismo de implementación de versiones lógicas (nivel lógico) y el nivel que trata la organización

física sobre la que se sustentan los procesos de Salva y Recuperación de versiones (nivel físico).

Operaciones definidas sobre el conjunto de versiones

1) Crear árbol de versiones
   Es la operación que se ejecuta al crear la versión cero. Inicializa una interfaz para la presentación del árbol de derivación.

2) Eliminar versión
   Elimina una versión sin descendientes. En caso contrario devuelve error.

3) Reestructurar árbol
   Es la operación designada para cortar el árbol en un número de versión específico. No se permite cortar la versión cero. El efecto de la operación de corte es convertir la versión donde se corta el árbol en versión cero, lo cual significa desechar las versiones antecesoras a ésta (Ver figura 4)

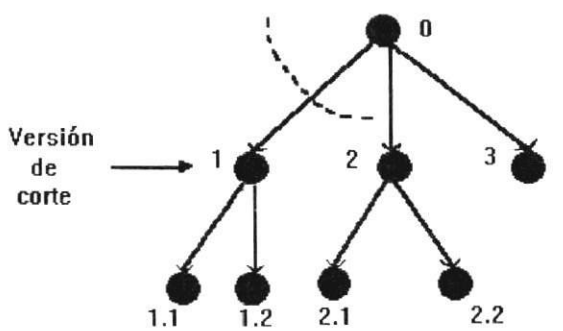

Fig 4- Ejecución de un «corte» en el árbol de derivación de versiones

En este ejemplo se desechan las versiones 0, 2, 2.1, 2.2 y 3. Al no interesar la versión cero, tampoco interesan sus derivadas. La versión 1 se convierte en la raíz del árbol de derivación.

La operación de corte permite a los diseñadores poder desechar versiones más viejas que ya no tengan sentido.

4) Funciones agregativas y de evaluación.

   Se definen estas funciones para operar sobre los valores numéricos en un conjunto de versiones obtenido. Se permitirá calcular la suma, promedio y hallar el máximo o mínimo valor entre los atributos del mismo nombre de una versión (a estas operaciones se les denomina «funciones agregativas» para una versión)
   También se podrán comparar varias versiones para hallar la que tiene el mínimo o el máximo valor de un atributo (funciones de evaluación)

5) Un lenguaje para la especificación de versiones

El propósito del citado lenguaje es permitir que el acceso a las versiones se haga no sólo utilizando un mecanismo navegacional sino por medio de un lenguaje de solicitudes que permita recuperar una o varias versiones de una vez. Según varios autores, esto es un propósito deseable (BER93), (HUGH91).

El lenguaje tiene una sintaxis parecida a SQL, aunque en las aplicaciones las solicitudes de versiones se definirán de manera interactiva, es decir, sin tener que teclear comandos.

Organización física

El modelo se basa en una organización física que puede manejar versiones de un objeto complejo de datos sin importar lo que éste represente lógicamente, en este caso, una

organización para poder implementar los procesos de Salva y Recuperación vistos en la figura 3.

La generación de los identificadores de versión en cada operación de creación formará implícitamente un árbol de derivación que representa una historia con alternativas según se vió en la figura 2. La versión inicial tendrá el identificador 0, las alternativas de ésta se enumerarán 1, 2, 3, etc. Las sucesoras de cualquier versión se identifican según la notación decimal Dewey.

Al crear una nueva versión, se almacenan en las versiones sólo los valores que cambiaron con respecto a la versión actual, llamémosle a este conjunto de valores *conjunto delta.* De esta forma cualquier versión requiere heredar de sus ancestras las propiedades que mantiene. Esto conduce a un ahorro en el espacio de almacenamiento.

Aplicaciones

Para validar el modelo, se han construído dos aplicaciones: un editor de páginas web y un software de ayuda al diseño de columnas de destilación.

Para fundamentar porqué en hipermedia se necesita el control de versiones, expondremos brevemente en qué consiste ésta y las particularidades del diseño de documentos utilizando esta técnica. Observaremos que la estructura de un documento hipermedia corresponde también a un objeto complejo de datos, y su actividad de diseño también requiere el uso de un modo conceptual de trabajo junto al trabajo en modo instancia.

Según Theodor Nelson (NEL87) *Hipermedia* es el término que define el almacenamiento y recuperación de información mediante la computadora de una manera no secuencial. Como extensión del término *Hipertexto* (escritura no secuencial), hipermedia implica enlaces y navegación en un material almacenado en cualquier medio, léase texto, video, sonido, gráficos, etc. La habilidad para moverse en la información textual y las imágenes es, no obstante, sólo la mitad del problema: un entorno que pueda denominarse con propiedad como hipermedia incluye herramientas que permiten al lector reelaborar el material que se le presenta con «control total del usuario».

Muchos autores consideran sinónimos *hipertexto* e *hipermedia*, mientras que otros consideran a la última como versión multimedia del primero.

La utilización de multimedia redunda en grandes beneficios cuando se trata de manejar grandes bases de datos de distintos medios. Los documentos voluminosos, las bases de datos con gran cantidad de referencias cruzadas o aquellas que han de ser consultadas por tipos de personas con intereses muy diferentes para recuperar información muy divergente, son candidatos claros a este medio.

Los sistemas hipermedia (una vez construídos) se caracterizan por dividirse en *unidades de información* que se denominan: frames, nodos, tópicos, etc. que pueden contener, además de información textual, gráficos, imágenes fijas, sonido y video.

Estas unidades de información se muestran una por ventana y se interconectan por *enlaces*, los cuales permiten la navegación. Los enlaces pueden definir interrelaciones estructurales (qué componente forma parte

de otro) y también pueden definir interrelaciones de asociación (estas útimas también llamadas «anotaciones»)

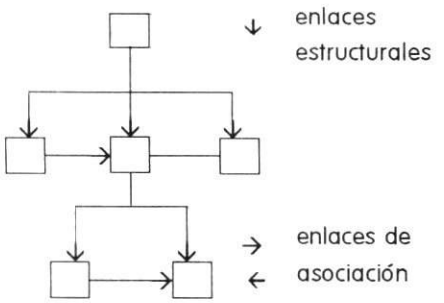

Fig. 5 - Enlaces entre los componentes de un hipertexto.

El texto forma el hilo conductor de la historia y comunica las ideas principales. Su formato define la estructura y jerarquía del contenido. Los enlaces permiten saltar tópico a tópico en cualquier momento y seguir referencias cruzadas, con lo que se facilita seleccionar y acceder a la información deseada. La búsqueda de información puede hacerse editando el frame seleccionado (lo que se denomina en inglés «browsing») o por búsqueda de cadenas de texto dentro de la jerarquía de frames. Se proporcionan también comandos para ir a un frame específico (go to).

Los sistemas de hipertexto presentan un formato de intercambio en una notación estándar denominada HTML (del inglés **H**yper**t**ext **M**arkUp **L**anguage) que en forma de script describe el contenido del hipertexto, aunque existen otros formatos.

Podría pensarse que puesto que los conceptos intuitivos que manejamos en hipermedia son fáciles de comprender y de utilizar por no expertos, la creación de un sistema hipertexto debe tener baja complejidad, y sin embargo no es esto lo que ocurre, se dan una serie de dificultades tanto para los autores como para los lectores de sistemas hipermedia, entre las que podemos citar que es bastante común perderse en el proceso de navegación. Lo que suele definirse como «pérdida en el hiperespacio» o en la «tela de araña» (ORT95), se caracteriza de varias maneras: el lector no sabe volver a un nodo que le interesaba, o se olvida de lo que estaba buscando, o cambia de objetivo en la búsqueda.

En un artículo (VAN87) Andries van Dam expresa: « ...En un sentido, los hipertextos brindan *go to*, y como sabemos, el *go to* produce spaghetti....Entonces necesitamos descubrir qué otras estructuras hay equivalentes.... Necesitamos nuevas formas, nuevos flujos de control....». Así mismo, define como « linkitis» los problemas que surgen durante la navegación.

En otro artículo más reciente (NAN95) aparece:

«Un ambiente de diseño de hipertextos debe integrar herramientas que no sólo faciliten y hagan eficiente el proceso de navegación, sino también que manipulen aspectos formales.... Una estructura de hipertexto debe ser descrita en términos de tipos lógicos. independientemente del contenido real de los nodos.... Para facilitar la creación de prototipos, los ambientes de trabajo deberán permitirle al diseñador *conceptualizar* una estructura general y ayudarle a instanciarla...»

De esta forma, trabajando en términos de instancias y en términos de abstracciones, se permitirá al diseñador que pueda alternar entre los procesos top-down y bottom-up del diseño.

En un artículo temprano (GAR88) se plantea que el filtrado de información en un hipertexto puede ser considerado

equivalente a definir visiones en una base de datos o usar un lenguaje de solicitudes para recuperar información de una base de datos, lo que hace pensar que tales mecanismos pueden ser útiles también para hipertextos. En su artículo «Reflexiones sobre NOTECARDS ™» (HAL88) Frank G. Halasz trata tendencias futuras de investigación en hipertextos entre las que cita al versionamiento

La implementación de COVER se llevó a cabo en Borland Delphi 2.0 y corre sobre **Windows '95** y **NT**. Necesita para ejecutarse un mínimo de 8 Mb de memoria, aunque es recomendable utilizar 12 o 16 Mb. La configuración mínima en que nuestro sistema ha sido probado es un 486 DX-2 a 66 MHz, con 8 Mb de RAM.

En la figura 6 se observa la interfaz de COVER, donde se aprecia una zona para definir el esquema del documento (en la pantalla «Schema») formado por una colección de componentes en la forma que el usuario los haya usado, una zona para de trabajar con el árbol de derivación de las versiones («Version»), y una ventana desplegada donde se observa la posibilidad de definir una solicitud para que COVER resalte los nodos-versión en el árbol de versiones que cumplen con los criterios de selección de la solicitud. En la tesis doctoral, base de este artículo, esta aplicación se describe más detalladamente y se muestra un ejemplo de versión de documento visto en Internet Explorer. Las versiones, por supuesto, admiten diferencias en la estructura para el documento, pues el usuario puede incluir nuevos componentes o eliminar algunos existentes, o cambiarlos de lugar entre una versión y otra.

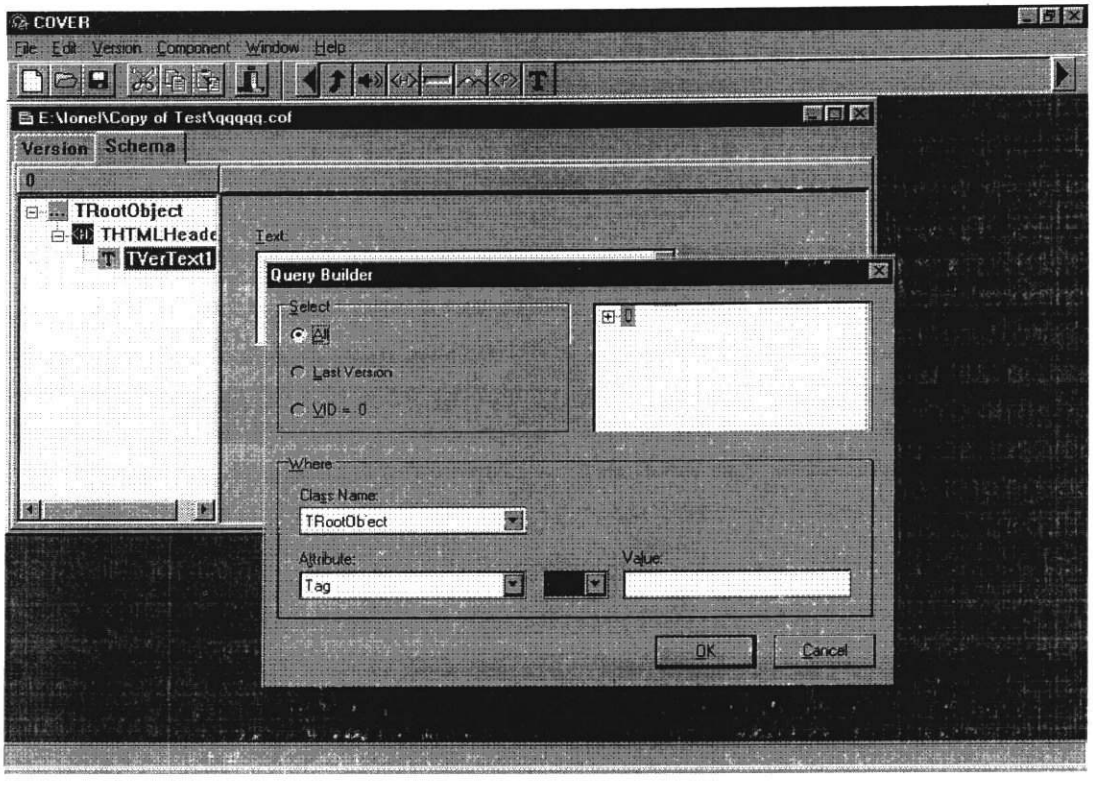

Figura 6 .- Interfaz del programa COVER.

El software de ayuda al diseño de columnas de destilación fue elaborado a partir de las bibliotecas de clases que para control de versiones fueron elaboradas con anterioridad.

CONCLUSIONES

Las ventajas que, en sentido externo, aporta este modelo para la creación de software de ayuda al diseño son las siguientes:

1. La incorporación del *modo de trabajo conceptual:* permite a los diseñadores definir un esquema (patrón) del objeto a diseñar, que resalta las relaciones estructurales entre los componentes del objeto, omitiendo los detalles del ensamble de las partes para ganar en abstracción. Este modo conceptual permite ser alternado con el trabajo en modo instancia.

2. La incorporación del manejo de versiones del objeto a diseñar *en el mismo ambiente de diseño,* lo cual incluye:

   - La creación de cuantas versiones admita el almacenamiento externo disponible
   - La recuperación de cualquier versión previa, admitiéndose el trabajo con versiones alternativas, no sólo revisiones lineales. La versión se recupera tanto en sus datos como en su esquema (es decir, las versiones pueden tener esquemas diferentes) y se permite la combinación de versiones de componentes para la creación de una nueva versión.
   - La posibilidad de usar un lenguaje para solicitar versiones del objeto que cumplan determinadas propiedades
   - La posibilidad de usar operadores aritméticos para elegir la versión que, dentro de un conjunto de versiones, satisfaga mejor ciertos criterios.

En sentido interno, el modelo propuesto se sustenta en una estrategia que minimiza el espacio necesario para el almacenamiento, ya que sólo se guardan los cambios de la versión actual con respecto a su ancestra directa; y la organización física basada en tablas permite que el lenguaje de solicitudes para versiones se pueda implementar con el apoyo de intérpretes SQL existentes.

El software obtenido permitió validar el modelo propuesto y ambas aplicaciones tienen valor práctico, además, las bibliotecas de clases implementadas permiten el desarrollo de aplicaciones en otras áreas.

Para hacer nuevos proyectos, se puede contactar con nuestro grupo de trabajo, de manera que:

1. El usuario puede desarrollar su propia aplicación suministrándosele las bibliotecas existentes

2. El usuario define el problema y nuestro grupo contrata la aplicación por completo.

REFERENCIAS

[AND94] ANDRÉ, Jöel y DELPECH, P. *Moving from Merise to Shlaer-Mellor.* Revista Objects in Europe. Verano de 1994, pp. 7-11.

[ANS91] *ANSI Database Language SQL.* Abril 1991. DIS 9075: 199X(E)

[BAT92] BATINI, C., CERI, S. y NAVATHE, S.B. *Conceptual Database Design.* Benjamin/Cummings Series on Database Systems and Applications, Redwood City, CA, 1992.

[BEE88] BEECH, D. y MAHBOD, B. *Generalized version control in an object-oriented database system.* En Proceedings of 4th. IEEE Int. Conference on Data Engineering, Los Angeles, CA., 1988.

[BLA88] BLAHA, M.R., PREMERLANI, W.J. y RUMBAUH, J. *Relational Database Design Using an Object-Oriented Methodology.* Commun. ACM, Vol.31, No.4, Abril, 1988, pp. 414-427.

[BER93] BERTINO, E. y MARTINO, L. *Object-Oriented Database Systems: concepts and architectures.* Addison-Wesley Publishing Co., 1993

[BET95] BETHONY, H. *Versions for Documents reins in oft-updated files.* PC Week, Marzo 20, 1995, 12(11)

[BIE95] BIEBER, M. y KACMAR, C. *Designing hypertext support for computational applications.* Communications of the ACM, 38 (8), Agosto 1995 pp. 99-107.

[BOO94] BOOCH. G. *Object Oriented Analysis and Design*, 2nd. ed. Benjamin Cummings, Redwood City, CA, 1994.

[BOR93] BORHANY, M. , BARTHES, J.P., ANOTA, P., GAILLARD, F. *A synthesis of the versioning problems in object-oriented engineering systems.* Proceedings of the IFIP, Compiegne, 1993

[COM94] COMMAFORD, C. *Version control is not optional, it's required.* PC Week, Nov. 7, 1994., 11(44).

[DATE94] DATE, C.J. *Oh Oh relational: toward an OO/Relational Rapprochment.* Database Programming & Design. Oct. 1994, 7(10) 23-27.

[DEL95] DELROSSI, R. A. *Version-control leader PVCS gets face-lift.* InfoWorld. Marzo 20, 17 (12), 1995.

[DEU91] DEUTSCH, P.L. *Object-Oriented Software Technology.* IEEE Computer 24(9), 1991 pp 112-113.

[DIJ94] DIJKSTRA, J. *On Complex Objects and Versioning in Complex Environments.* Proc. IFIP; Compeigne, 1994

[GAR88] GARG, P. K. *Abstractions mechanisms in hypertext.* Communications of the ACM , 31 (7), Julio 1988 pp. 862-870.

[GAR95] GARCÍA, A. *Diseño de una solución para el manejo de versiones en SIADI.* Reporte Técnico, UCLV, Cuba, 1995

[GARZ95] GARZOTTO,F., MAINETTI,L. y PAOLINI, P. *Hypermedia design, analysis and evaluation issues.* Communications of the ACM, Agosto de 1995 38 (8), pp. 74-86.

[GOD95] GODDARD, D. *Object Database Technology: What's ahead?* DataBased Advisor, Nov. 1995 pp. 64-69.

[GON94] GONZÁLEZ, L. *SIADI: Sistema Integrado de Ayuda al Diseño.* Tesis en Opción al grado de Doctor en Ciencias Técnicas. Universidad Central de Las Villas, Cuba, 1994

[HUGH91] HUGHES, J.G. *Object-Oriented Databases.* Prentice-Hall. C.A.R. Hoare series ed., 1991

[JAC92] JACOBSON, I. , CHRISTERSON, M., JOHNSON, P. y ÖVERGAARD, G. *Object-Oriented Software Engineering: A use case driven approach.* Addison-Wesley, 1992.

[KATZ90] KATZ, R. *Toward a unified framework for version modelling in engineering databases.* ACM Computing Surveys, 22(4), 1990 pp.375-408.

[KIM89] KIM W., BALLOU, N., CHOU, H. T., GARZA, J. and WOELK, D. *Features of the ORION object-oriented database system.* In Object-Orient Concepts, Databases and Applications. (Kim,W. and Lochovsky,F., eds.) pp 251-282. Reading, MA:Addison-Wesley, 1989.

[KLA86] KLAHOLD, P., SCHLAGETER, G. y WILKES, W. *A general model for version management in databases.* En Proceedings of the International Conference on VLDB, Kyoto, Japón, Agosto 1986.

[KOR95] KORTH, H.F. y SILBERSCHATZ, A. *Fundamentos de bases de datos.* Segunda Edición, Mc Graw-Hill, Méjico, 1995.

[LAN86] LANDIS, G. S. *Design evolution and history in an O-O CAD/CAM Database.* En Proceedings of the 31th. COMPCON Conference, San Francisco, CA. Marzo 1986.

[LEACH95] LEACH, N. *Intersolv MicroSoft expand version control.* PC-Week. Marzo 6, 1995 12 (9) pp. 27-29.

[MIC95]. MicroSoft Systems Journal, Enero 1995. 10 (1). Version Control (Software Packages)

[MUÑ96] MUÑOZ, I. *Sistema autor de documentos multimedia con control de versiones.* Tesis de Maestría en Computación Aplicada. Universidad Central de Las Villas, Julio de 1996.

[NAN95] NANARD, J. y NANARD, M. *Hypertext design environments and the hypertext design process.* Communications of the ACM Agosto 1995 38 (8) pp. 49-56.

[ORT95] ORTEGA, M., RUIZ, F., BRAVO, J., RUÍZ, J. y DOMÍNGUEZ, E. *Hipermedia.* Conferencia Internacional sobre Informática Educativa, Universidad de Oviedo, España, 1995.

[PRES93] PRESSMAN, R.S. *Ingeniería del Software: un enfoque práctico.* 3ra. ed. Mc Graw-Hill, Méjico, 1993.

[ROD94] RODDICK, J. F., CRASKE, N.G., RICHARDS, T. *A taxonomy for schema versioning based on the Relational and Entity-Relationship models.* Proc. IFIP, Compiegne, 1994.

[RUM91] RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F. y LORENSEN, W. *Object-Oriented Modelling and Design.* Englewood Cliffs, N.J: Prentice Hall, 1991.

[SCIO91] SCIORE, E. *Multidimensional Versioning for object-oriented databases.* 2nd. International Conference DOOD´91. Münich, Germany. Dic. 16-18, 1991 Proceedings, Springer-Verlag.

[SEE94] SEER, K. y WISE, M. *A framework for managing model objects.* Database Programming & Design, Agosto, 1994 7 (8).

[SHN87] SHNEIDERMAN, B. *Designing the user interface: strategies for effective human-computer interaction.* Addison-Wesley Publishing, Co. Reading Mass, 1987.

[SUN88] Sun Microsystems. *Introduction to the NSE.* SUN Part No. 800-2362-1300 (Mar 7 1988)

[STO90] STONE, C.M. y HENTCHEL, D. *Database Wars Revisited.* BYTE, Oct. 1990 pp. 233-242.

[VIN88] VINES, P. , VINES , D. y KING, T. *Configuration and Change Control in GAIA.* ACM, New York, 1988.

[WARD94] WARD, P. *The role of objects in multimedia.* Revista «Objects in Europe» Vol. 1 No. 3 Verano 1994.