UN AMBIENTE DE DESARROLLO DE SOFTWARE PARA EDUCACION

Pedro Hepp. K., Ph.D.
Jaime Navón C., M.Sc.
Escuela de Ingeniería
Universidad Católica de Chile
Casilla 6177 (143) - Santiago

Se propone un conjunto de componentes de software que simplifican la generación de aplicaciones en apoyo de la educación, en especial en el diseño de la interfaz con el usuario. Se muestra la utilidad de estas componentes en la construcción de un laboratorio de estructuras de datos en el cual los alumnos pueden observar el comportamiento de diversas estructuras de datos en situaciones de carga real.

1. INTRODUCCION

La utilización del computador como herramienta de apoyo a la educación ha tenido un significativo auge con la aparición de los microcomputadores, haciendo posible su utilización en apoyo a la docencia en colegios y universidades y a la capacitación en empresas.

El desarrollo de la educación asistida por computador (CAI, Computer Aided Instruction) ha permitido explorar el uso de diversos sistemas puestos a disposición de los profesores para construir sus aplicaciones. Ejemplos son los proyectos Plato de la universidad de Illinois y Ticcit de la corporación Mitre,y más recientemente los lenguajes Smalltalk [Gold 83]y Logo [Pape 80]. Estos sistemas han tenido diversos grados de aceptación entre los profesores quienes tienden a preferir la mayor flexibilidad que les permiten los lenguajes de propósito general, tales como Basic, Pascal y Logo para construir sus propios programas [Niev 86].

Los rápidos avances de la tecnología computacional aumentan dia a dia el potencial de uso de los microcomputadores en educación. Nuevos procesadores de gran velocidad y poder, memorias de varios millones de bytes, pantallas de alta capacidad gráfica que ofrecen la posibilidad de utilizar colores, ventanas, íconos y video interactivo. A esto se suman los avances en psicología cognitiva [Hulm 85] [Reid 85], nuevas ideas en torno a tutores inteligentes [Slee 82], etc.

Sin embargo, esta tecnología hace cada vez más compleja la programación de las aplicaciones, haciendo necesario el conocimiento de técnicas de Ingeniería de Software para obtener programas que sean modificables, robustos y que utilicen adecuadamente los recursos computacionales. Es así como a los profesores con conocimientos básicos de programación les resulta cada vez más difícil aprovechar los avances de la tecnología y deben contentarse con aplicaciones más modestas o bien utilizar programas de generación de aplicaciones CAI, tales como Pilot con los cuales pierden parte de la flexibilidad de un lenguaje de programación.

Puesto que el costo del diseño de la interfaz con el usuario y su implementación representan típicamente más de un 50% del costo total de desarrollar una aplicación [Shoo 83] [Somm 85] y que otros componentes, tales como el manejo de la persistencia representan cerca de un 30% [Atki 83], resulta evidente que éstas son las áreas en que se requiere mayor apoyo.

En este trabajo presentamos una alternativa al profesor con conocimientos básicos de programación. Ofrecemos un ambiente de desarrollo de software en el cual existe una gran facilidad para diseñar las componentes de la interfaz hombre-máquina, automatizando la generación de código a partir del diseño y proporcionandole componentes de software reusables que manejen la persistencia de los datos.

A continuación se presentae el ambiente de software, se muestran algunos ejemplos de diseño

y finalmente se desarrolla una aplicación que implementa un laboratorio de estructuras de datos, destacando la ventajas pedagógicas y las componentes de la interfaz con el usuario.

2. HERRAMIENTAS Y COMPONENTES DE SOFTWARE

Como se ha mencionado, los profesores prefieren la utilización de lenguajes de propósito general debido a la total flexibilidad que les permite, pero esto dificulta la programación de aplicaciones que aprovechen las nuevas tecnologias. Además, se hace cada vez más difícil modificar las aplicaciones para mantención y principalmente ante cambios en el diseño.

Nuestro enfoque se basa en la premisa que es suficiente (y deseable) proporcionar flexibilidad al nivel de diseño de una aplicación, particularmente en el diseño de la interfaz con el usuario, sin ser necesario dejar al profesor acceso a todo el código. Este enfoque permite automatizar la generación de parte del código a partir del diseño, acortando significativamente el costo y tiempo de producir una aplicación. Un enfoque similar es utilizado por el sistema ScriptWriter [Horo 87]

2.1 Herramientas de Software

El software que se utiliza para diseñar la interfaz con el usuario, lo denominamos **herramienta**. Una herramienta es un programa interactivo con el cual un profesor interactúa para especificar su diseño. Una vez especificado, la herramienta genera código para implementar el diseño. En el código generado se incluye en uno o más módulos de software que son posteriormente acoplados a la aplicación.

Algunos ejemplos de herramientas de software útiles para el desarrollo de aplicaciones CAI son las que se utilizan para: la construcción de menús tipo cortina; la definición de ventanas a fin de poder manejar áreas individuales de la pantalla como si fuesen pantallas completas; el diseño de íconos para construir figuar y especificar su desplazamiento en las pantallas; la definición de diálogos para simplificar el proceso de ingreso de opciones y datos por parte del usuario al interactuar con la aplicación , etc.

Este enfoque tiene diversas ventajas. En primer lugar permite la generación rápida de aplicaciones, con un mínimo de funcionalidad, pero con una interfaz completa, a fin de que los usuarios experimenten con ésta y propongan los cambios necesarios. La interfaz es fácilmente rediseñada, generando nuevos módulos con código y el profesor puede ir agregando paulatinamente nuevas funciones. En segundo lugar, la generación de código automático disminuye los errores, ya que éstos son poco a poco eliminados del módulo correspondiente en cada herramienta a medida que son reusadas. En tercer lugar, las herramientas facilitan al profesor la construcción de diversas aplicaciones con cierta estandarización de la interfaz con el usuario, disminuyendo para éstos el costo de aprendizaje de cada nueva aplicación del profesor.

2.2 Componentes de Software

Junto con las herramientas, que generan modulos de software a partir de un diseño, se encuentran las **componentes** de software. Una componente es un módulo de software que cumple una función específica, tiene una clara definición para facilitar su incorporación a una aplicación, pero no es modificable sino por medio de cambios en su implementación. A diferencia de las herramientas, los componentes son "cajas negras", normalmente no modificables.

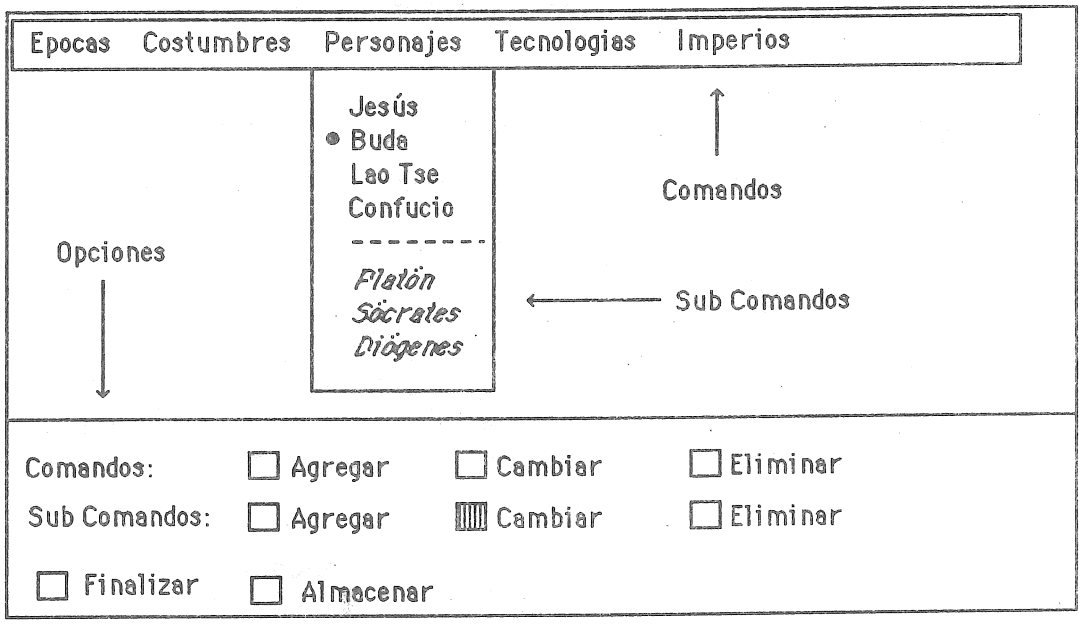
2.2.3 Componentes Específicas de la Aplicación.

En esta categoría entra el resto del código necesario para construir la aplicación. Normalmente incluye software de enlace para unir las distintas componentes que hayan sido incorporadas de los dos primeros grupos, y software específico que resuelve problemas muy particulares y que por lo tanto no puede ser considerado en la segunda categoría. En la medida que la base de software de los dos primeros grupos sea suficientemente completa, el trabajo necesario para hacer la aplicación será reducido.

2.3 Ejemplo de Herramienta: Menús de tipo Cortina

En el caso de diseño de menús de tipo cortina, la herramienta permite especificar la ubicación y contenido de cada uno de los menús. El código generado contiene dos tipos de módulos de software. Un grupo de módulos que maneja los menús activados por una mouse o a través de teclas y otro grupo de módulos que contiene un esqueleto de programa que debe ser completado por el profesor con el código que implementa los comandos de su menú. Parte de éste código es generado automáticamente como es el caso de los comandos para abrir y cerrar archivos, almacenar el contenido de ventanas de texto y gráficas, etc.

La siguiente figura muestra un ejemplo de diseño de menús de una aplicación cualquiera:



El profesor desarrolla su diseño a través de opciones activadas con la mouse o con teclas. Por razones de espacio, en la figura no se muestran otras opciones, como por ejemplo, para indicar si un subcomando está activo en cierto momento, para separar grupos de subcomandos, etc.

Típicamente, en cualquier aplicación orientada a CAI, se distinguen 3 grandes categorías de componentes de software :

- componentes de software generales de CAI
- componentes de software del dominio de la aplicación
- componentes específicas de la aplicación

Dado que un objetivo importante es que la aplicación pueda ser desarrollada por las mismas personas que estarán involucradas en su uso, es decir los profesores, es necesario separar conceptualmente estas componentes y en lo posible tratar de reducir al mínimo las del último tipo. Lo ideal es que el profesor encuentre, entre las componentes del primero y segundo tipo todos los elementos para construir un porcentaje significativo de la aplicación.

2.2.1 Componentes de Software Generales de CAI.

Dentro de esta categoría se encuentran todas las componentes básicas del ambiente de apoyo al desarrollo de aplicaciones. Ejemplos típicos de este tipo de componente son aquellas que se requieren en apoyo de los módulos generados por las herramientas como el manejador de mouse y el manejadore de íconos.

Otras componentes son el manejador de gráficos, editor de propósito general, el manejador de la persistencia de los datos, componentes matemáticos, estadísticos y de monitoreo.

El manejador de mouse permite controlar las acciones que el usuario indica con este dispositivo, como por ejemplo, al seleccionar un ítem en la barra de menús. El manejador de íconos permite interactuar con las pequeñas figuras o íconos que han sido incorporados a la aplicación, por ejemplo en lugar de poner un texto que diga que se está trabajando con un árbol binario, es preferible mostrar el ícono correspondiente.

El manejador de gráficos permite mostrar gráficamente los valores especificados en forma tabular e incluye diversos tipos de gráficos, tales como barras, tortas y líneas y formatos de presentación. El editor de propósito general permite manejar todo el ingreso de texto en forma simple y homogénea, e incluye operaciones de cortar, pegar, etc. El manejador de la persistencia de los datos administra almacenamiento y recuperación de información desde ventanas de texto y gráficas. Finalmente, la componente de monitoreo permite almacenar el trabajo efectuado por cada usuario con la aplicación a fin de que el profesor pueda analizarlo.

2.2.2 Componentes de Software del Dominio de la Aplicación.

Son las componentes que están íntimamente ligadas al tipo de aplicación que se está construyendo, pero que podrían ser reutilizadas en otras aplicaciones del mismo tipo. Por ejemplo, si el dominio es el de las estructuras de datos, podrían tenerse componentes como: lenguaje algorítmico, manejador de objetos gráficos , etc. El lenguaje algorítmico permitiría especificar en un lenguaje de muy alto nivel (similar al pseudo-código usado en los textos), un cierto algoritmo. El programa interpretaría este código para ejecutarlo directamente, pudiendo así mostrar en forma paralela este código simplificado, y lo que ocurre con los datos. El manejador de objetos gráficos simplificaría la tarea de manipular los objetos involucrados en una animación.

3. CASO DE ESTUDIO: UN LABORATORIO DE ESTRUCTURAS DE DATOS.

3.1 Motivación.

La capacidad de estimación aproximada, o de orden de magnitud, de lo bueno o malo que resulta una cierta estructura de datos, es ciertamente difícil de inculcar a los estudiantes en un curso semestral de estructuras de datos [Navo 86] [Navo 87].

Por ejemplo, al tomar un caso tan simple como un árbol de búsqueda binario, difícilmente podrán contestar "a priori" la altura que tendrá un árbol de 10.000 nodos. Se les enseña que la altura, o número de accesos es una función logarítmica del número de nodos (O(log n)). Sin embargo, también se les dice que en el peor caso se llega a una estructura de lista (árboles degenerados). Al calcular el rango de variabilidad para la altura el estudiante obtiene que el número de accesos puede ser desde 14 ($\log_2$ 10000) hasta 10.000 ! La pregunta inmediata es ¿y en un caso práctico cuánto será ?

Una ayuda muy importante en la superación de este tipo de dificultados puede lograrse mediante la generación de un ambiente experimental, donde el alumno pueda observar, y comparar el comportamiento de las estructuras de datos mas conocidas frente a las operaciones básicas. En realidad, casi las mismas razones que hacen aconsejable el uso de laboratorios para aprender Física o Química se mantienen en este caso [Balk 85] [Boot 86].

La idea es efectuar experimentos con volúmenes grandes de datos y poder confrontar así los resultados experimentales con los teóricos aprendidos en las clases. El alumno, despues de la clase teórica sobre árboles binarios se dirige al laboratorio de estructuras (microcomputadores equipados con el software adecuado), y realiza experimentos reales generando árboles con 1.000, 100.000 o 1.000.000 de elementos. Podrá observar entonces, la altura de ellos cuando las llaves son escogidas en forma random o con alguna otra distribución en particular y obtener de este modo una mejor comprensión de cómo se comporta un árbol de búsqueda binario en la realidad.

3.2 Requerimientos Básicos.

Para satisfacer los objetivos anteriores, el software debe satisfacer algunos requisitos fundamentales :

1) Facilidad de Diseño de Laboratorios "Ad Hoc".

El instructor debe poder construir o modificar un ambiente de laboratorio de acuerdo a sus propios objetivos. Lo que se desea es un minilaboratorio a la medida, de acuerdo a los objetivos que se plantee el instructor para la unidad correspondiente.

Por ejemplo, si se están enseñando estructuras de árboles balanceados, el instructor podría "armar" un laboratorio que incluya árboles AVL, árboles 2-3 y árboles binarios simples como referencia para las comparaciones. No sólo las estructuras son piezas intercambiables sino también distribuciones de datos, operaciones y observación de resultados.

2) Interfaz gráfica autoexplicativa.

El alumno debiera poder operar el programa casi sin necesidad de apoyo de documentación

auxiliar (manuales), sólamente basado en lo que observa en pantalla [Böck 86] [Niev 86]. Esto implica una clara preferencia al uso de interfaces gráficas, "pull down menus", ventanas, etc, frente a interfaces de tipo comando en base a texto.

3) Agilidad del ciclo preparación - prueba - observación.

Si queremos que el alumno explore y experimente, debe poder hacerlo con facilidad. Un ejemplo resulta clarificador de este punto. Supongamos que el alumno está investigando el comportamiento de las tablas de hashing para lo cual carga inicialmente la tabla al 80% y luego realiza búsquedas observando el número de accesos requerido. El impulso inmediato es probar que pasaría con factores de carga iniciales diferentes, pero el alumno solo lo hará si eso no le significa especificar nuevamente todos los parámetros para posteriormente esperar 10 minutos por los nuevos resultados. Si por el contrario, el ciclo es ágil, en unos minutos habrá probado como afecta esta variable al desempeño de esta estructura.

4) Volúmenes de Datos grandes.

Una de las motivaciones fundamentales para hacer un experimento de este tipo con el computador, es el poder "poblar" las estructuras con volúmenes de datos acorde a la realidad, es decir, grandes. En contraposición a los ejemplos triviales presentados en clases, i.e., árboles con 4 a 8 nodos, el alumno puede ahora insertar 10.000 o mas nodos en el árbol lo cual es obviamente imposible de hacer sin esta herramienta.

5) Observación gráfica del desempeño.

En lo posible, representar en forma gráfica los resultados de los experimentos en lugar de tabular números (una imagen vale mas que mil palabras). Esto contribuye tambien a agilizar el ciclo, de acuerdo a lo planteado en el tercer requerimiento.

6) Reforzamiento de Conceptos.

Despues de observar el desempeño de la estructura, el alumno debe sacar algunas conclusiones de lo que vió. En este momento debe apoyarse con el conocimiento teórico que hay tras el experimento realizado mediante una precisa exposición del comportamiento esperado.

3.3 Componentes del Software.

3.3.1 Componentes de Software Generales de CAI.

En esta categoría tenemos las componentes que ya han sido mencionadas en el punto anterior:

- manejador de mouse
- manejador de gráficos
- manejador de la persistencia
- editor de propósito general
- constructor de íconos

y los componentes generados por las herramientas para manejar menús, ventanas, diálogos e íconos.

3.3.2 Componentes de Software del Dominio de la Aplicación.

En este caso, el dominio es el de las estructuras de datos, y las componentes necesarias:

- estructuras
- operaciones
- datos
- distribuciones
- observaciones

Las estructuras son las distintas construcciones con las cuales será posible trabajar: árboles, listas, stacks, AVL's, B-trees, etc. Las operaciones se refieren a las distintas cosas que se podrá hacer sobre los datos de las estructura: insertar, buscar, eliminar, rotar, dividir, mezclar, etc. La componente datos permite seleccionar el tipo de dato que se maneja al interior de las estructuras: enteros, reales, strings, dibujos, etc. Las distribuciones, muy relacionadas con los datos, permiten extraer modos de generación de datos en volúmen: uniforme, normal, secuencias, etc. Las observaciones se refieren mas bien al tipo de variable que se observará como resultado de los experimentos: número de accesos, número de comparaciones, número de llamadas de procedimientos, etc.

3.3.3 Componentes Específicas de la Aplicación.

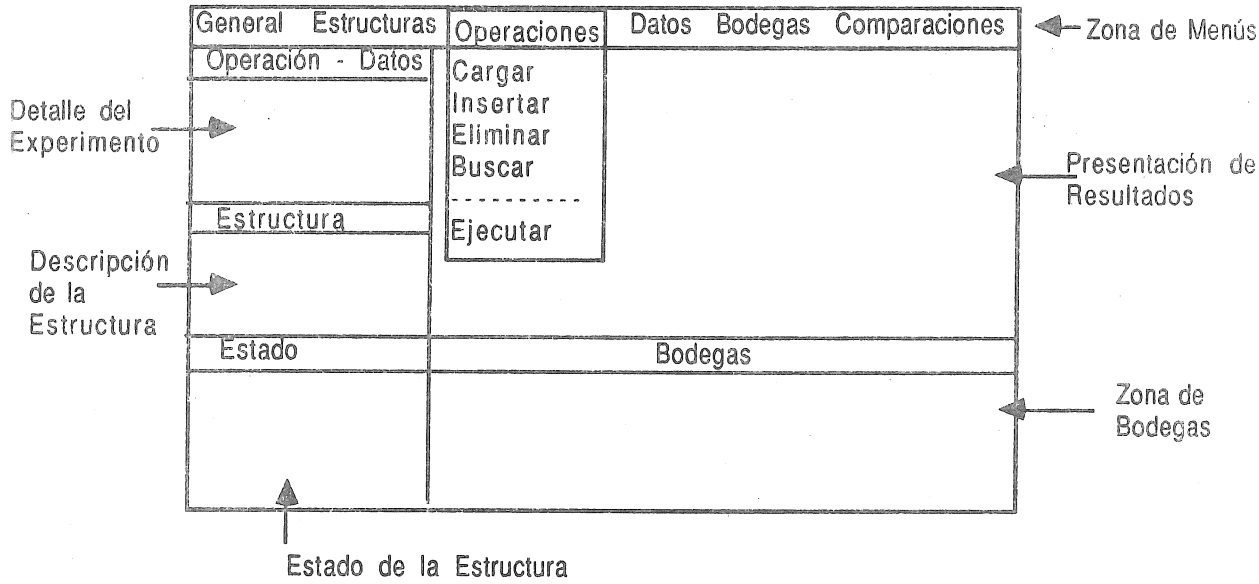
No detallaremos aquí este tipo de componentes por ser muy particular de la aplicación. Corresponde al resto del código que permite construir la aplicación que se describe.

3.4 El Laboratorio.

Supondremos que hemos efectuado ya la etapa de "armado" del laboratorio para lo cual seleccionamos de entre las componentes descritas en el punto anterior lo siguiente :

Estructuras - B-trees y tablas de hashing
Operaciones - Buscar, insertar y eliminar (en ambos casos lo mismo)
Datos - números enteros
Distribuciones - uniforme y secuencial (ordenada)
Observaciones - Número de accesos

Luego de completada la etapa de armado, obtenemos un módulo ejecutable que contiene el ambiente de laboratorio que el instructor ha seleccionado para sus alumnos. Al activarlo vemos una pantalla como lo ilustra la figura.



Existen en esta pantalla 6 ventanas o áreas fundamentales :

1. La zona de los menús.-
   Desde aquí se controlan todas las acciones a seguir. En este caso se definieron 6 menús (utilizando la herramienta descrita anteriormente)

   i) General - Contiene comandos de ayuda (help), comandos para "guardar" el laboratorio en el estado en que se encuentra para uso posterior, comandos para cargar un laboratorio que ha sido guardado, y comandos para terminar el programa.

   ii)Estructuras - Contiene las distintas estructuras con las cuales se puede experimentar. En el estado actual este menú tiene dos items : árboles binarios y tablas de hashing.

   iii) Operaciones - Contiene las operaciones básicas que se efectúan sobre la estructura: insertar, borrar y buscar, ademas de un comando para cargar inicialmente la estructura. Tenemos en este menú tambien el comando para comenzar la simulación.

   iv) Datos - Todo lo relacionado con el dominio y número de datos además de la distribución que siguen los datos que están siendo insertados (uniforme, normal, secuencial).

   v) Bodegas - Permite hacer el manejo comparativo entre varios experimentos relacionados entre sí por el cambio en algún parámetro, o bien entre estructuras distintas con el mismo experimento.

   vi) Comparaciones - Permite controlar la observación de los resultados (presentación es en forma gráfica).

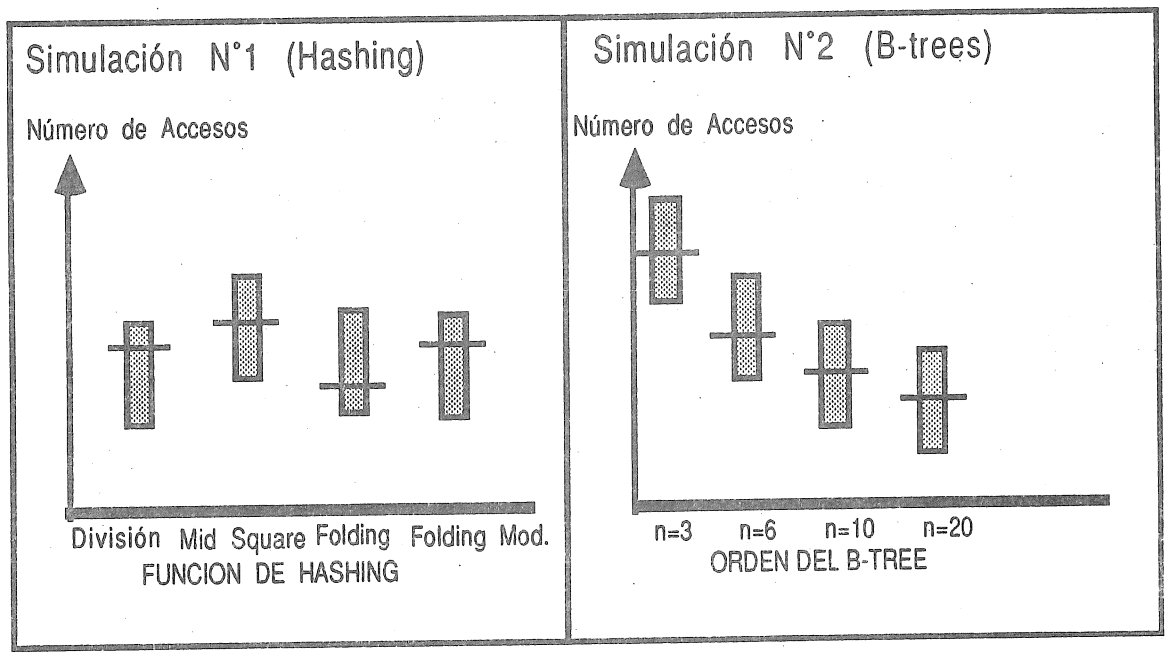
2. La zona del detalle del experimento.
   Aquí aparece el detalle del experimento o simulación que se está efectuando.

3. La zona de descripción de la estructura.
   Aquí aparece el detalle de la estructura que está siendo estudiada. Nombre de la estructura y todos los parámetros que definen la instancia del experimento.

4. La zona del estado de la estructura.
   Aquí se encuentra el detalle del estado de la estructura en un cierto instante. Por ejemplo si se trata de árboles binarios, el número de llaves, la altura, etc.

5. La zona de bodegas.
   Cuando se desea comparar el desempeño de dos o mas estructuras, se puede almacenar el resultado de un experimento o simulación en las llamadas "bodegas" y repetir la simulación cambiando la estructura. El área de bodegas ilustra el estado de las bodegas, es decir los distintos resultados que podríamos analizar comparativamente.

6. La zona de resultados.
   Es el sector donde se observa el resultado de la simulación efectuada. Es posible observar gráficamente el desempeño de las estructuras: cómo varía al aumentar el volumen de datos o frente a cambios en algunos parámetros (ver figura).



4. CONCLUSIONES

El ambiente de desarrollo de software presentado ofrece un conjunto de herramientas de diseño y componentes de software para la construcción rápida de aplicaciones CAI. Este software ha sido probado para armar laboratorios de estructuras de datos que prueban la utilidad de este enfoque. Se espera ampliar el potencial de las herramientas de diseño para aumentar el grado de automatización en la generación de software educacional.

AGRADECIMIENTOS

Este trabajo ha sido realizado gracias al apoyo de la Dirección de Investigación de la Pontificia Universidad Católica de Chile (DIUC, Proyectos 15/1987 y 22/1987), del Consejo Nacional de Ciencia y Tecnología (Proyecto 757) y de IBM de Chile.

5. REFERENCIAS

[Atki 83] ATKINSON, M.P., BAILEY,P.J., CHISHOLM,K.J., COCKSHOTT,W.P., MORRISON, R. (1983) An Approach to Persistent Programming. En **PS-Algol Papers**. University of Edinburgh, Department of Computer Science, PPR-2-83.

[Balk 85] Balkovich E., Lerman S., Parmelee R. (1985) Computing in Higher Education: The Athena Experience". **Communications ACM Vol 28** , Nro 11, Nov , 1214-1224.

[Böck 86] Böcker Heinz-Dieter, Fisher Gerard, Nieper Helga (1986) The Enhancement of Understanding through Visual Representations. **Conference Proceedings on Computer and Human Interaction.** CHI 86 Abril,6 44-50.

[Boot 86] Booth T. et al. (1986) Design Education in Computer Science and Engineering **Computer**, Junio, 20-27.

[Gold 83] GOLDBERG, A., ROBSON, D. (1983) **Smalltalk-80. The Language and its Implementation**. Addison Wesley, Reading.

[Horo 87] HOROWITZ, E. (1987) **ScriptWriter Reference Manual**. Computer Science Department, University of Southern California, Los Angeles, Ca.

[Hulm 85] HULME, C. (1985) Reading: Extracting Information from Printed and Electronically Presented Text. En **Fundamentals of Human-Computer Interaction**, Andrew Monk (Ed.), Academic Press.

[Navo 86] NAVON J. (1986) Aprendizaje de Estructuras de Datos mediante la Experimentación" **II Encuentro Nacional de Educación y Computación**, Viña del Mar, Chile.

[Navo 87] NAVON J. (1987) Basis for a Data Structures Laboratory **11th Educational Computing Conference**, San Francisco, 19-20 Nov iembre.

[Niev 86] NIEVERGELT, J., VENTURA, A., HINTERBERGER, H. (1986) **Interactive Computer Programs for Education. Philosophy, Techniques and Examples**. Addison-Wesley, Reading.

[Pape 80] PAPERT, S. **Mindstorms: Children, Computers and Powerful Ideas.** Basic Books, New York, 1980.

[Reid 85] REID, P. (1985) Work Station Design, Activities and Display Techniques. En **Fundamentals of Human-Computer Interaction.** Andrew Monk (Ed.), Academic Press.

[Shoo 83] SCHOOMAN, M. (1983) **Software Engineering. Design, Reliability and Management**. McGraw-Hill, Nueva York.

[Slee 82] SLEEMAN, D., BROWN, J.S. (Eds.) (1982) **Intelligent Tutoring Systems**. Academic Press, London.

[Somm 85] SOMMERVILLE, I. (1985) **Software Engineering**. 2ª edición, Addison-Wesley, Wokingham.