INTERFASES DE USUARIO ORIENTADAS A OBJETOS. UN ENFOQUE COMPARATIVO

Autores:Aguirre, G.C. Errecalde, M.L. *

Grupo de Interés en Sistemas de Computación**

Departamento de Informática - Universidad Nacional de San Luis

Ejército de los Andes 950 - Box 106

Cod.Postal 5700 - San Luis - Argentina

Tel:(0652) 20823

e-mail: gaguirre@unsl.edu.ar y merreca@unsl.edu.ar

RESUMEN

Una de las áreas en que la programación orientada a objetos produjo una verdadera revolución y posibilitó una amplia difusión del paradigma, es la del diseño de interfases de usuario. Algunos autores llegan a considerar como sinónimos los términos "interfases de usuario orientadas a objetos" y las hoy tan populares "interfases icónicas".

En este contexto surgen distintas arquitecturas para el diseño general de este tipo de interfases las cuales si bien coinciden en los lineamientos generales, presenta cada una sus propias características.

Este trabajo compara los aspectos principales de tres de ellas, el paradigma MVC(Modelo-Visión-Controlador), la propuesta de Cox para interfases de usuario icónicas y el esquema adoptado en el paquete Turbo Visión para C++, poniéndose énfasis en dos conceptos troncales en este tipo de interfases como son el de supervisión/subvisión y el manejo de eventos.

PALABRAS CLAVES

Interfases orientadas a objetos, visiones, eventos.

---

* Miembros del proyecto 338403 de la UNSL.

** El grupo de investigación es subvencionado por la Universidad Nacional de San Luis, el CONICET (Consejo Nacional de Investigaciones Científicas y Técnicas) y la SSID (Subsecretaría de Informática y Desarrollo).

1. INTRODUCCIÓN

En los últimos tiempos, los especialistas en computación se han encontrado con un nuevo fenómeno: el uso masivo de computadoras personales por un número creciente de usuarios, en su mayoría inexpertos.

Es así que el objetivo de hacer más fácil de usar a las computadoras y en particular las interfases hombre maquina, comienzan a tornarse un factor crítico, caracterizado por la búsqueda permanente de amigabilidad con el usuario.

Las tradicionales interfases orientadas a comandos, caracterizadas por una interacción con el usuario basada en el ingreso de comandos mediante el teclado y la visualización de la información como meros números y palabras, comienzan a tornarse obsoletas. En su lugar, se hacen cada vez más importantes, interfases que intentan presentar al usuario las tareas computacionales de una manera más cercana a sus actividades cotidianas.

Es así, como surgen interfases con ventanas y figuras solapadas que crean al usuario un ámbito de trabajo similar al que se presenta en el escritorio de su oficina (metáfora de desktop), y donde la selección de sus elementos de trabajo ahora se efectúa mediante algún dispositivo de señalamiento como el mouse. En este ámbito, por ejemplo, un usuario no necesitará conocer la sintaxis de un comando para eliminar un documento, sino que bastará con seleccionarlo con el mouse y moverlo al icono que representa al cesto de papeles.

El diseño de este tipo de interfases tuvieron como base en su gran mayoría al paradigma orientado a objetos y más específicamente las ideas surgidas del paradigma MVC (Modelo-Visión-Controlador) que sirvió como base para el diseño de la interfase de usuario del sistema SMALLTALK.

El concepto de encapsulamiento posibilitó, en estos casos, controlar la complejidad intrínseca de este tipo de interfases, mientras que la herencia permitió que a partir de componentes genéricas (clases) ya provistas con los sistemas de programación orientados a objetos, se construyeran en forma sencilla y segura interfases de usuario para aplicaciones particulares.

Este trabajo presenta en primer lugar las principales características del paradigma MVC, analizándose los puntos en común y diferencias con la propuesta de Cox[3, 4] de interfases de usuario icónicas. Posteriormente se describen algunos inconvenientes que se presentan en la implementación de la metáfora del "desktop" y como el concepto de jerarquía supervisión/subvisión cumple un rol fundamental en la solución de estos inconvenientes.

2. ARQUITECTURAS DE INTERFASES ORIENTADAS A OBJETOS

Sin lugar a dudas el paradigma MVC (Modelo-Visión-Controlador), estableció un nuevo esquema para la construcción de interfases de usuario, que influenció en mayor o menor medida la mayoría de las interfases de usuario posteriores.

Básicamente, consiste en dividir la interfase de usuario en tres tipos de objetos:

- *Modelo*: Representa la estructura de datos de la aplicación. Contiene o tiene acceso a la información que será mostrada en sus visiones.
- *Visión:* Maneja todas las tareas gráficas; requiere datos del modelo y los muestra. Una visión puede contener **subvisiones** y estar contenidas en **supervisiones**. Esta jerarquía de *supervisiones/subvisiones*, es la que posibilita manejar múltiples ventanas, moverlas o redimensionarlas.
- *Controlador:* Provee la interfase entre su modelo/visión asociados, y la entrada del usuario. Adicionalmente interactúa con otros controladores.

Estas 3 componentes, están interconectadas como se muestra en la figura siguiente:

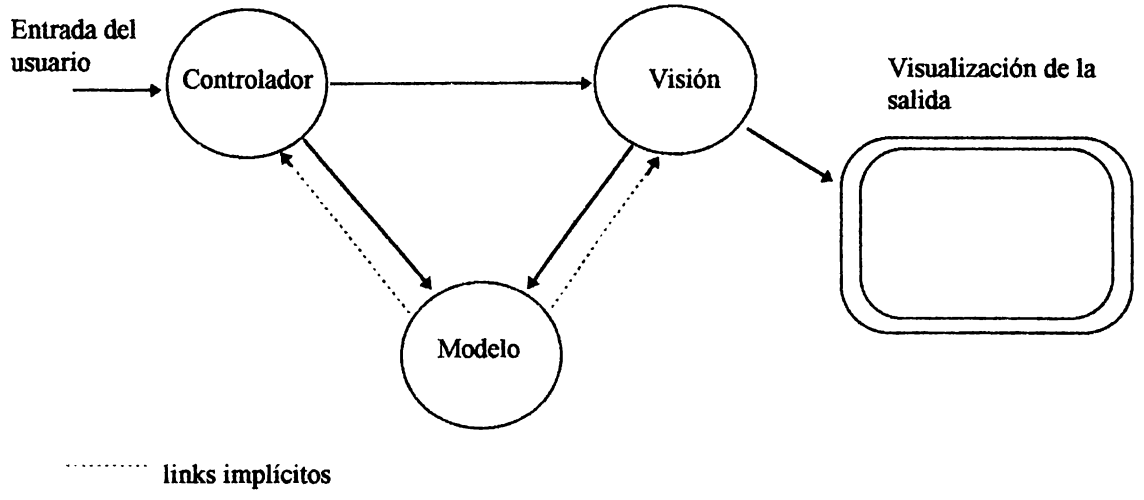

Figura 1: Esquema MVC

En este esquema, las acciones de entrada del usuario son manejadas por el controlador activo, el cual cumple el rol de objeto interfase, respondiendo mediante la invocación de las acciones apropiadas en el modelo. El modelo realiza la operación solicitada (posiblemente modificando su estado), e informando a todas sus visiones asociadas (por medio de links implícitos), que él ha cambiado. Cada visión puede luego consultar al modelo por su nuevo estado y actualiza, si es necesario, la manera en que se muestra.

En SMALLTALK, el esquema MVC es implementado mediante 3 superclases (llamémosle Modelo, Visión, Controlador), y numerosas subclases de éstas, que soportan las interfases particulares.

La mayoría de los modelos, tienen un par Visión/Controlador asociado. Sin embargo, el modelo no tiene acceso directo a las visiones y controladores. La única

interacción desde el modelo hacia las visiones y controladores, es para notificar cuando él ha cambiado su estado.

Si bien el paradigma MVC, no fue discutido en su conjunto en ninguno de los 3 libros principales escritos por los creadores de SMALLTALK, sus ideas fueron explotadas en ambientes de desarrollo como de los de Lisa y Mackintosh, y dieron origen a nuevas arquitecturas como las propuestas por Cox y Yen-Ping[7].

A continuación se describirán las características generales de la propuesta de Cox para interfases de usuario icónicas.

En la propuesta de Cox, al igual que en el paradigma MVC, intervienen modelos y visiones que trabajan conjuntamente para interactuar con el usuario.

El modelo mantiene la información que debe ser presentada al usuario, y las visiones constituyen la interfase propiamente dicha.

En esta arquitectura, tanto los modelos como las visiones se agrupan en capas, llamándolas capas de **Aplicación** y **Presentación** respectivamente.

Los objetos (**modelos**) que componen el nivel de Aplicación, implementan la funcionalidad de una aplicación particular, mientras que los objetos (**visiones**) que componen el nivel de Presentación, constituyen las distintas interfases de usuario, que hacen referencia a los modelos de la aplicación.

Algunas visiones de la capa de Presentación serán genéricas, y otras específicas de la aplicación.

Una visión debe cumplir distintas funciones: obtener datos del modelo, determinar el formato de los datos, presentar información al usuario, recibir comandos del usuario y ordenar al modelo que hacer.

La relación entre modelo y visión es tal, que únicamente la visión puede mantener punteros al modelo, pero nunca será el modelo quien referencie a la visión. En la gráfica siguiente, se visualiza esta situación, mostrando a la aplicación como "esclava" de la interfase de usuario.

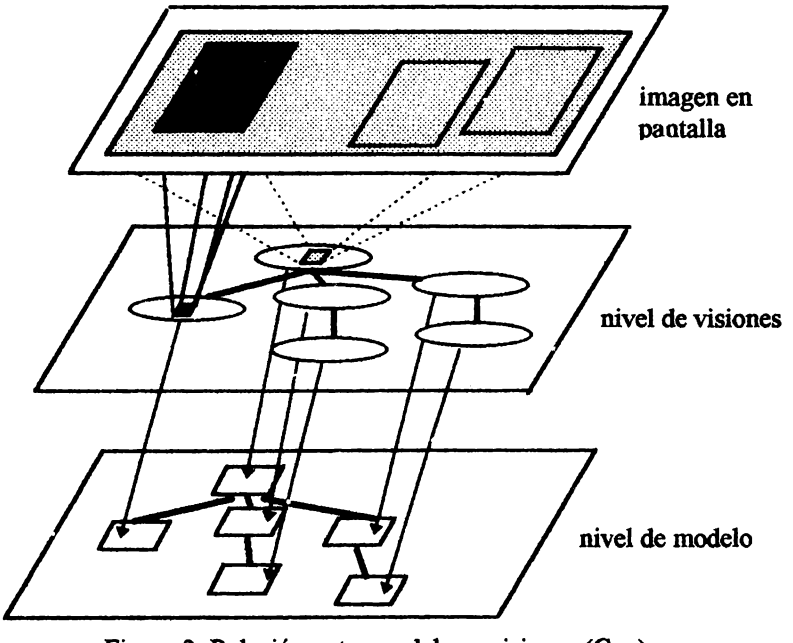

Figura 2: Relación entre modelos y visiones (Cox)

Los modelo son totalmente pasivos y no mantienen información que pueda hacerlos dependientes de ninguna interfase particular.

Esto es importante ya que posibilita que se reemplace la capa de Presentación, sin que la aplicación se vea afectada.

Cox considera además una tercera capa denominada **nivel de Terminal Virtual**, el cual consiste básicamente en una biblioteca de primitivas gráficas para la construcción de figuras elementales (líneas, rectángulos, etc.) en forma independiente del dispositivo gráfico a usar.*

Como se puede observar la propuesta de Cox es muy similar a la usada en SMALLTALK con la excepción que en esta última la capa de Presentación consiste no de una sino de dos jerarquías de clases (Visión y Controlador). Cox considera que la ventaja de tener una jerarquía de controladores separada no es muy clara (ni aun en la comunidad SMALLTALK), mientras que el hecho de permitir que cada visión soporte una interfase de usuario completa, hace al diseño de la interfase más fácil de describir y entender.

Salvando estas diferencias, existen características principales comunes a ambos enfoques:

1. Existe una jerarquía de visiones, formada por visiones genéricas, a partir de las cuales se definen clases que implementan visiones particulares específicas de la aplicación.
2. Los objetos de la capa de Presentación (visiones), referencian a los objetos de la capa de Aplicación (modelos), pero nunca a la inversa.
3. El usuario interactúa con la aplicación mediante el concepto de control dirigido por eventos.

Con respecto a punto 1, la organización usual, consiste en definir una clase (llamémosle **Visión),** al tope de esta jerarquía, cuyas instancias entre otras cosas responden a los mensajes de "dibújese", "cambie de tamaño", "muévase". Normalmente no tendrá sentido crear objetos de esta clase, ya que ésta constituye una clase abstracta que implementa el comportamiento de una hoja de acetato transparente.

Todas aquellas clases declaradas como subclase de **Visión**, heredarán esta funcionalidad y le agregarán aquellas características propias del tipo de visión que implementan (visión de menú pull-down, caja de diálogo, ventana de edición, etc.).

La figura siguiente, muestra una jerarquía parcial de tipos de visiones , que ejemplifica este tipo de organización:

*El nivel de terminal virtual, no será discutido aquí debido a la carencia actual de un consenso general sobre un stándard gráfico.

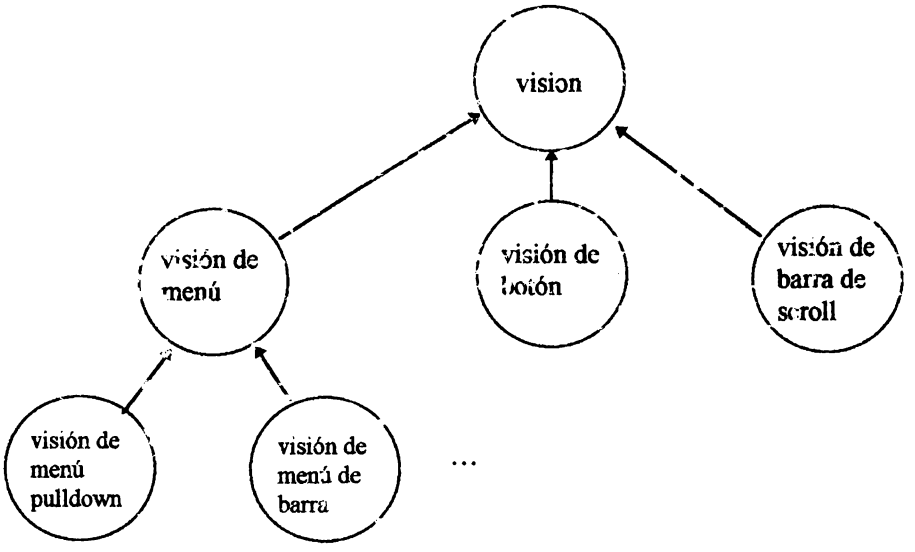

Figura 3: .Jeraraquía de clases de visiones

Este tipo de situaciones, es un buen ejemplo de la utilidad de la herencia en el paradigma. Además de la obvia reducción en el esfuerzo de programación al reusar métodos definidos en las clases superiores, la jerarquía de clases de visiones se adapta perfectamente en aquellas situaciones donde se requiere polimorfismo.

Un ejemplo de esto es la operación "**dibújese**" de una visión no primitiva (aquellas formadas por otras visiones, por ejemplo una caja de diálogo se compone de líneas de entradas, botones de radio, botón de aceptar, cancelar, etc.). Esta operación asume que sus visiones componentes son del tipo genérico **Visión,** y que cada una sabe responder al mensaje **"dibújese".** En este caso, se le envía a cada visión componente, la cual será una instancia de alguna subclase de visión, el mensaje **"dibújese",** ejecutándose el método correspondiente de cada visión particular.

Esta jerarquía base de clases de visiones, servirá de soporte para definir visiones que son propias de la interfase a implementar. Esto permite que, además de una reducción substancial en el esfuerzo de programación, todas las aplicaciones que utilizan esta jerarquía exhiban una interfase de usuario consistente.

Con respecto al punto 3, al hablar de eventos se los puede considerar como ocurrencias aleatorias que se generan por distintos motivos y a los que la aplicación debe responder. Los eventos más conocidos son los que se generan por acción del mouse, del teclado o los generados por las mismas visiones para comunicarse entre ellas.

A diferencia de las interfases tradicionales, una aplicación dirigida por eventos no debe preocuparse por aceptar la entrada del usuario, debido a que existe un soporte que provee esta funcionalidad y que además genera un evento el cual sí será responsabilidad de la aplicación manejar.

En la sección siguiente, se explicará el concepto de jerarquía supervisión/subvisión el cual es sumamente importante para entender como se organizan las visiones no primitivas, no solo en cuanto a su presentación en pantalla sino también en la manera en que responden a diferentes clases de eventos.

3. JERARQUÍA SUPERVISIÓN/SUBVISIÓN

Uno de los problemas principales de las interfases orientadas a ventanas consiste en la manera de manejar un número arbitrario de visiones, que pueden cambiar de tamaño y posición y superponerse entre sí. Esto es consecuencia de simular en el plano un efecto tridimensional, lo que trae aparejado una dificultad en el modo en que las visiones se dibujan en pantalla y en la determinación de cual es la que debe manejar un determinado evento, por ejemplo el clic de un botón de mouse por parte del usuario.

La forma usual de resolver estos problemas, es organizar las visiones de manera tal, que se permita que una visión sea dueña o esté compuesta de otras visiones.

Es así, que una visión puede ser insertada o agregada a otra visión, formándose de esta manera un árbol de visiones, en donde cada visión es el telón de fondo (background) de sus subvisiones.

La relación entre la visión de background y aquellas asignadas inmediatamente a ella es implementada mediante una estructura de datos denominada **jerarquía supervisión/subvisión.**

En la propuesta de Cox, toda visión es asignada a otra que provee su background (su supervisión) y puede potencialmente servir de background a otras visiones (sus subvisiones) Estas relaciones son administradas mediante dos variables de instancia de la clase View: **superView** y **subViews**.

Cuando una visión es creada, su variable subViews es inicializada como una colección en la cual se almacenarán sus subvisiones Si la visión es terminal, es decir que no está compuesta por subvisiones, esta colección estará vacía.

Otras plataformas, como Turbo Vision, marcan una clara diferencia entre visiones terminales y aquellas visiones formadas por otras visiones, a las que llama **visiones complejas.**

Todas las clases que implementan visiones complejas, están agrupadas bajo una clase común denominada **TGroup.** La jerarquía parcial de clases para visiones utilizada por Turbo Vision se muestra en la figura siguiente:

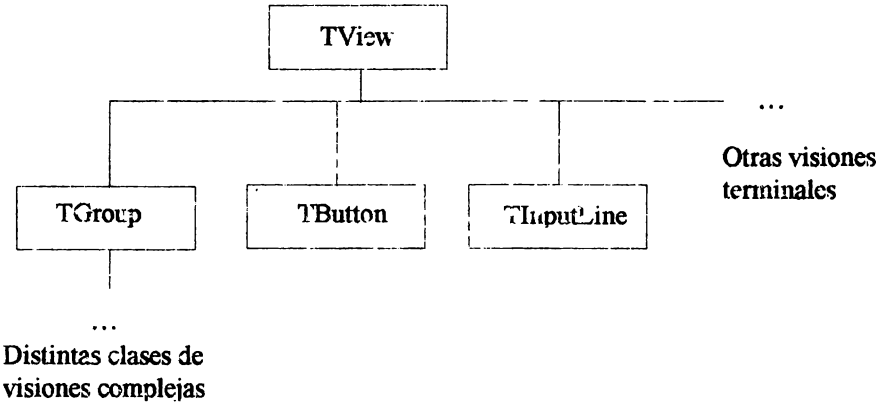

Figura 4:Jerarquía parcial de visiones de Turbo Vision

Básicamente un grupo es sólo una caja vacía que contiene y administra otras visiones, implementando entre otras cosas la funcionalidad para la inserción, supresión y visualización en pantalla de las visiones que la componen.

Si bien a semejanza de Cox, todas las visiones mantienen una referencia a su supervisión o **visión dueña** (en este caso un objeto TGroup), sólo TGroup y no TView, es la clase que tiene consciencia de las subvisiones en su conjunto.

La jerarquía supervisión/subvisión en cualquiera de los dos casos analizados, provee el soporte para implementar los aspectos tridimensionales de la metáfora del escritorio, manteniendo información del orden en que las visiones han sido insertadas.

Este orden, referenciado en Turbo Vision bajo el nombre de **orden-Z**, es el que permite resolver la situación de visiones solapadas, determinando el orden en que son dibujadas y el orden en el que son manejados, y por cual de ellas, los eventos generados por el usuario .

Con este ordenamiento se logra el efecto de que la última visión insertada se muestre como la más cercana al usuario. Para ello se aplica el siguiente procedimiento general: cuando una visión compleja debe mostrarse en pantalla, lo primero que hace es mostrar todas las imágenes visuales que le son propias y que servirán de fondo para todas sus subvisiones. Luego solicita a sus subvisiones que se muestren en pantalla. Dado que estas subvisiones pueden a su vez ser visiones complejas, este procedimiento se repetirá hasta que se muestren las visiones terminales.

Esta política de trabajar desde las visiones que se encuentran más atrás hacia las de adelante, no sólo puede ser aplicada para dibujarlas, sino también para determinar como manejar los eventos.

En una aplicación dirigida por eventos, existen básicamente tres etapas:

1. Detección del evento.
2. Ruteo del evento.
3. Manejo del evento, propiamente dicho.

El punto uno consiste generalmente en un ciclo en el cual la aplicación espera hasta que un evento ocurra. Cuando se detecta la ocurrencia de un evento se procede a realizar la acción denominada ruteo del evento. Para el caso particular de un evento de mouse, esto consistirá básicamente en ubicar cual es la visión al tope en el lugar donde se hizo el clic del mouse y derivarle el evento para que lo maneje.

Para realizar este cometido, tanto la propuesta de Cox como la de Turbo Vision utilizan la jerarquía supervisión/subvisión de la siguiente manera: el evento es dirigido en primer lugar a la visión más alejada del usuario. Esta visión controla si las coordenadas donde se produjo el clic del mouse caen dentro del área ocupada por la visión. En caso que así sea dirige el evento a todas sus visiones componentes.

Este proceso se repite hasta que una visión (la más cercana al usuario) no tenga subvisiones a las cuales derivar el evento, y por consiguiente deberá hacerse cargo de su manejo.

La solicitud para el manejo del evento se realizará mediante un mensaje particular. Turbo Vision difiere con la propuesta de Cox en que el primero tiene un único mensaje para el manejo del evento (**handleEvent**) el cual hace la discriminación de que evento particular se trata, mientras que Cox propone transformar el evento en un mensaje particular a ser enviado a la misma visión, siendo responsabilidad de ésta definir un método por cada uno de los mensajes que puede manejar (**leftButtonDown**, **rightButtonDown**, etc.).

El enfoque de Cox, con respecto a este último punto, el cual puede ser considerado como más puramente orientado a objeto, podría justificarse en que el lenguaje sobre el que se implementa su interfase (Objective-C), permite el envío de mensajes cuyos nombres se conocen en tiempo de ejecución, utilizando para ello mensajes del tipo *perform*. Esto permite convertir el nombre de un evento en un mensaje particular, lo que no es posible en forma automática en C++.

4. CONCLUSIONES

Distintas arquitecturas de interfase de usuario han sido analizadas y el principal aspecto en que difieren es en el hecho de tener o no una jerarquía separada de controladores, siendo éste un tema de discusión aún en la comunidad SMALLTALK.

Salvando estas diferencias, existen características principales, comunes en general a todos los enfoques:

1. Existe una jerarquía de visiones, formada por visiones genéricas, a partir de las cuales se definen clases que implementan visiones particulares, específicas de la aplicación.
2. Los objetos de la capa de Presentación (visiones), referencian a los objetos de la capa de Aplicación (modelos), pero nunca a la inversa.
3. El usuario interactúa con la aplicación mediante el concepto de control dirigido por eventos.

Este tipo de interfases trabajan en general con la metáfora de desktop, lo que implica simular en el plano un efecto tridimensional. En este sentido cumple un rol fundamental el concepto de jerarquía supervisión/subvisión, no solo en cuanto a la presentación en pantalla de las visiones sino también en la manera en que responden a diferentes clases de eventos.

5. BIBLIOGRAFÍA

[1] Aguirre, G.C., Errecalde M.L. -SHOOE: Un shell orientado a objetos para Sistemas Operativos. Anales 21 Jaiio, 1992, Págs. 3.71-3.82.

[2] Aguirre, G.C., Errecalde M.L. -Una Arquitectura de interfase de usuario, estructurada en capas y basada en el paradigma orientado a objetos. Anales 23 Jaiio, 1994, Págs. 6.25-6.36.

[3] Cox B.J. -Object-Oriented Programming: An Evolutionary Approach. Addison-Wesley, Reading, Massachusett, 1986.

[4] Cox B.J., Hunt Bill -Objects, icons and software-IC's - Object Oriented Computing . Vol.2:Implementations - IEEE Computer Society Press (1987) - Pág 99-108.

[5] Goldbert A. and Robson D. -Smalltalk-80 The Language and its implementation. Adisson Wesley, Reading, Ma, 1983.

[6] Turbo Vision for C++. User's Guide. Borland International Inc. (1991)

[7] Yen-Ping Shan - MoDE: A UIMS for Smalltalk. ECOOP/OOPSLA '90 Proceedings. - October 1990 - Págs 258-268.