Transparent API

API Transparente

Luis Garreta[1] y Sandra Roa[2]
1. Universidad del Valle, Colombia.
2. Universidad del Cauca, Colombia
lgarreta@univalle.edu.co; smroa@unicauca.edu.co

Recibido para revisión 11 de Marzo de 2008, Aceptado 19 de Mayo de 2008, Versión final 23 de Mayo de 2008

Resumen— Este artículo presenta un enfoque para un API Transparente basado en Lentes Mágicas. Los lentes mágicos son filtros móviles que se desplazan sobre un área de trabajo de una aplicación y muestran nuevas vistas de los objetos sobre los que se superponen. El API Transparente propuesto es un mecanismo de visualización que se caracteriza por: no obstruir la visión de los objetos del área de trabajo, uso de elementos comunes de interacción con el usuario de interfaces tradicionales, proceso activo de datos, y la posibilidad de usarlo como un elemento adicional en alguna interfaz grafica de usuario (GUI).

*Palabras Clave:*Interfaz Gráfica de Usuario, Lentes Mágicas, Interfaz de Usuario Transparente.

Abstract—This paper describes an approach for a Transparent API (Application Programming Interface) based in Magic LensTM. Magic Lenses are movable filters that displace over an application work area and show new views of objects viewed through that area. The transparent API proposed is a visualization mechanism that have tools distinguished by: unobstructed view of the objects in the working area, use of typical interaction elements with user of traditional interfaces, active process of data, and the possibility to use it as an additional element in any graphic user interface (GUI).

Keywords: Graphic User Interface, Magic Lenses, Transparent User Interface.

I. INTRODUCTION

The bases for development of the transparent interfaces was made for Xerox at its Xerox-Parc Laboratory. Some of the features include: the transparent interface does not prevent the direct visualization over working applications, neither it takes permanent screen space; its tools move over working objects and can be configured in a way that shows new views of the objects [1].

The transparent API includes Magic Lens and ToolGlassTM widgets, that are used in an application work area. Magic Lens shows modified views of objects viewed through it. Instead, ToolGlass widgets float over the application work area without interfering with direct views of the objects place underneath the ToolGlass, and also they show options to interact with the user.

In spite of the advantages, the transparent interface and its tools are not common in the real GUIs yet [8] [3]. One of the reasons is that changing from screen windows toward transparent interfaces is a step too complex, instead, the introduction of some tools from transparent interfaces could be done easily in progressive steps.

Our focus in this paper is the introduction of some of these tools as an active visualization and interaction mean inside real graphic interfaces. As part of this work, we have built an application that use Magic Lens as visualization tools and a transparent API to interact with the user. The API includes typical elements of an interface, such as buttons, toolbars, checkboxes, etc.

This paper is organized as follows. The next section describes the transparent interface and its operation. The following section describes the basic principles of Magic Lens operation and the last section describes the transparent API including tools interaction, some details of its implementation and use inside an application. Finally we present our conclusions and reference material.

II. TRANSPARENT INTERFACES

The See-Through interface is defined as a set of semi-transparent interactive tools that appear on a virtual sheet of transparent glass called a Toolglass sheet, between the application and a traditional cursor. The set of tools that belongs to the see-through interface is called Toolglass widgets. These widgets are used over an application work area and may provide a customized view of the application underneath them, using viewing filters called Magic Lens filters. [1] [9]

A. Transparent Interface Operating

Basic operation of the interface begins when the user positions a toolglass sheet over a desired object in an application and then points with a mouse through the widget that appear in the toolglass sheet, letting the widget perform its operation. Results are presented in the same widget area and they are the outcome of applying a filter or an operation configured in advance over the objects placed underneath the widget.

The toolglass sheet placed between screen and application intercepts the user's inputs modifying them and giving them to the application. Likewise, the toolglass sheet intercepts display requests from the application, modifying them and showing them in the user screen. [2]

When an operation is accomplished in an interface tool, four steps are followed: trigger, movement, action, and appearance. The user triggers an event; the tool specifies an action based on the event and performs that action; when the action is performed an appearance is fixed to display the tool and the application underneath it; finally, the tool moves over the application objects and in each movement a new appearance of the objects is shown through the tool selected.

B. Transparent Interface Operating

Magic Lenses constitutes an interface tool that combine an arbitrarily shaped region with an operator (filter) that changes the view of objects viewed through that region [5]. These tools employ an attractive metaphor based on physical lenses; the user select a lens, he places the lens over the application area and he sees the new view that is produced by the lens [4]. The context is kept, in other words, the region outside the lens is kept unmodified, also lenses overlap showing composition of their effects. Figure 1 shows a part of the map from the city of Cali, Colombia. Over the map we have placed two lenses that show detail from the city blocks. The user can move the lens over any region, at each movement the lens dynamically gets the block information and shows it in the same region of the lens. The context outside the lens is kept unmodified, in this way we can see the block information without losing sight of the districts around the lens.

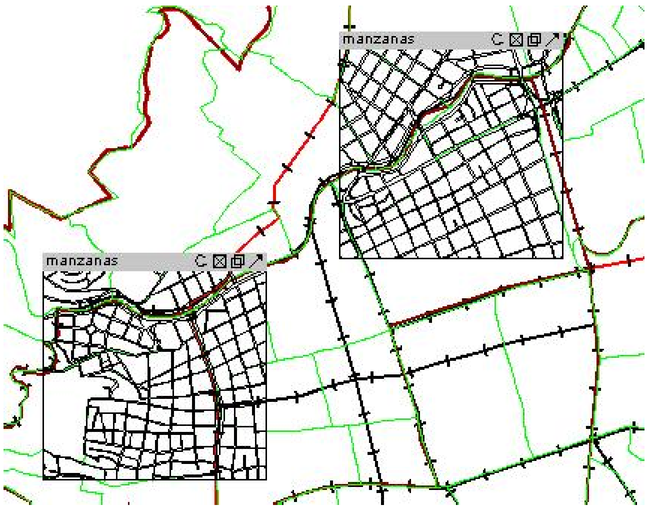

Figure 1. Lenses to view blocks in a Cali, Colombia area

III. TRANSPARENT API

The transparent API proposed has a set of Transparent Tools (TT) that are interface elements placed over an application work area that keep the object view underneath them without changing. The TT introduce elements to the user in order that he may interact with the application, such elements are buttons, menus, toolbars, etc. Also, the TT allow the user to interact with the objects through the tools, that means, the tools float over the application and the user interacts with the objects behind the TT as if they were not present.

Figure 2 shows two TT in form of toolbars: the upper one presents options to be performed over the map, the right one allows the user to check or uncheck the layers that conform the map such as district, municipality, cycloroute, perimeter, blocks and geological aspects of the region.

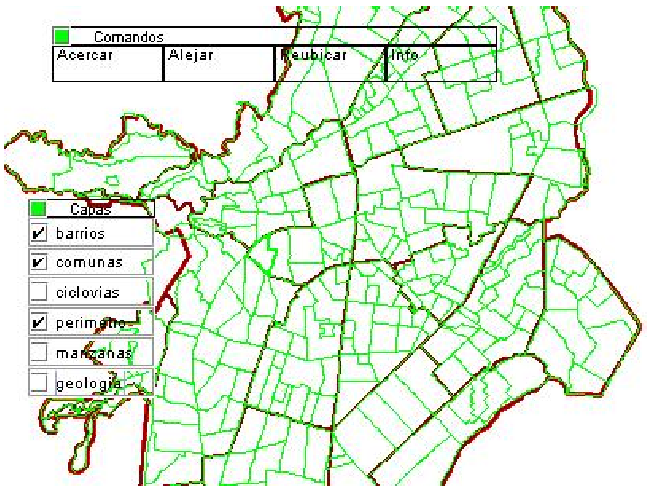

Figure 2. Transparent Tools with options to apply over the map

A. Interaction with the tools from the transparent API

Provided that one of the principal objectives of the transparent tools is to allow the interaction -in a transparent way- with the objects placed underneath the tools, then, the way to access them change slightly. Once we access a TT, the interaction with the TT is similar to the way we interact with the traditional tools.

Two hands can be used to operate a See-Through Interface [1]. Each hand operates a different input device, the dominant hand positions a cursor from a mouse while at the same time the non-dominant hand uses a trackball. Thus, the TT could be accessed by the trackball that the non-dominant hand could move it, while with the other hand -operating the mouse we could perform individual actions that are inside the tool [7]. Although, this two-handed operation works well with the s-through interface, in the real window interfaces this method is not the appropriate one, because the principal input devices - and most of time the only one - is a mouse.

For that matter, we proposed an alternative way that use only the mouse to access and perform actions inside the TT. For that, we use the concept of the two-handed operation from the see-through interface translated to the buttons in the mouse. If the mouse have two buttons, it is use as follows: to access the

TT we press the alternate button of the mouse (usually the right one) and to perform an action we press the principal button (usually the left one).

Therefore, if a user does not press the alternate button to access the TT and he presses the principal button in the mouse, the actions are performed over the objects placed underneath the tool and they are not executed over the elements that TT contain.

B. Architecture of the transparent tools

The TT that appears models under the architecture MVCL (Model-View-Controller-Lenses). In this architecture the Model represents the data of the the problem domain; the Vista presents/displays graph or textually to the Model; the lenses are located on the Views and change the appearance of the objects located in her; finally the Controller, coordinates to the previous elements, centralizes the communication and handles the logic of the application[6].

In the MVCL, the actions they are captured by the interaction elements that offer to the views and the lenses, if these actions deal with the logic of the application, then the element sends them controller to its so that it makes them. All this communications diagram becomes by means of the contract mechanism, where an element declares the actions that can capture but that it cannot solve them. The controller who is going to handle to the element, must fulfill the contract.

C. Implementation of the transparent API

The Transparent API proposed to be used as a TT, was implemented using JAVA as a programming language. The transparent API uses the most high level classes of Abstract Window Toolkit (AWT) that came with the JAVA language and from the components SWING are derived.

Figure 3 shows a diagram of one part of the TT Transparent API that contains the transparent toolbar options (BarraOpcionesTransparente). This class contains one or more transparent buttons (BotonTransparente) that constitute the available options. The class declares a contract (ContratoBarraOpcionesTransparente) with the options that can get from the common buttons: (MouseClicked) and with the options that can get from the checkboxes (opcionSeleccio nada,opcionDeseleccionada).

The high level class Container offers the basic structure to accomplish an object container and above all, it allows the possibility to overwrite its display method paint in such a way that when this method is executed only the outline is painted, in other words, text and background are not painted. Likewise happens with the high level class Component, the method paint is overwritten to paint outline and text only.

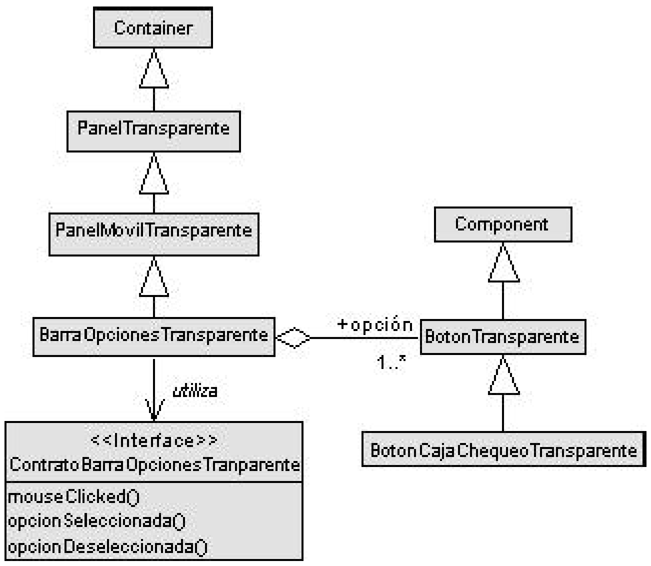

Figure 3. Diagram of classes for the element BarraOpcionesTransparente.

Figure 4 shows a part of the Class Diagram in UML that presents the options toolbar inside a GUI. The options toolbar declares in a contract the actions that capture but not resolve. Instead, the controller that guides the options toolbar accomplish (implement) the actions. Otherwise, the options toolbar contains a reference to the controller through a reference in the contract that it declares.

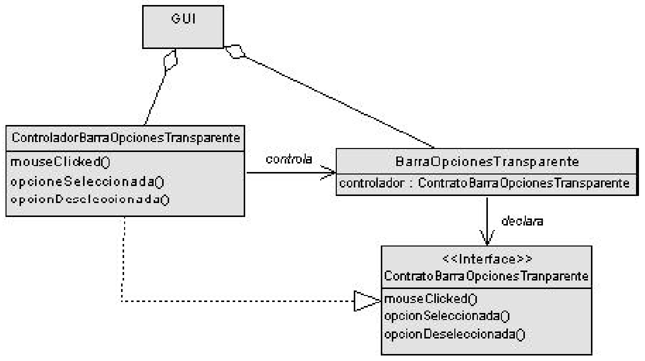

Figure 4. Transparent options toolbar inside a GUI.

D. Display of maps using the transparent API

An outline of an application that use the transparent API is shown in Figure 5. In this figure appears four views, the view to select the layers, the view to select the commands (Zoom in, Zoom out, etc.), the view to select the lenses and the view to display the maps. Note that lenses are applied over the maps view.

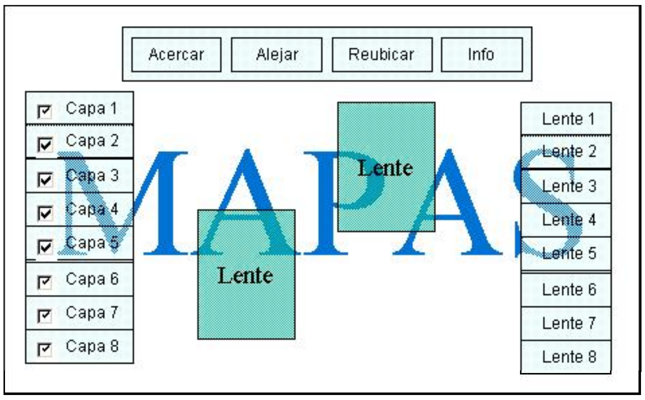

Figure 5. Outline of the transparent interface to view maps

Figure 6 shows the transparent API with the magic lenses interacting as elements of active visualization to display multiples and simultaneous floating views inside the same interface. Note that not all of the interacting elements are now transparent (element placed at the left of figure), this is other property from the transparent API. Additionally, the transparent API allows the user to make visible the interacting elements he wishes to make it appears, define the transparency level of the interacting elements, and to manipulate properties of lenses such as: sizes, processing ways and initial conditions.

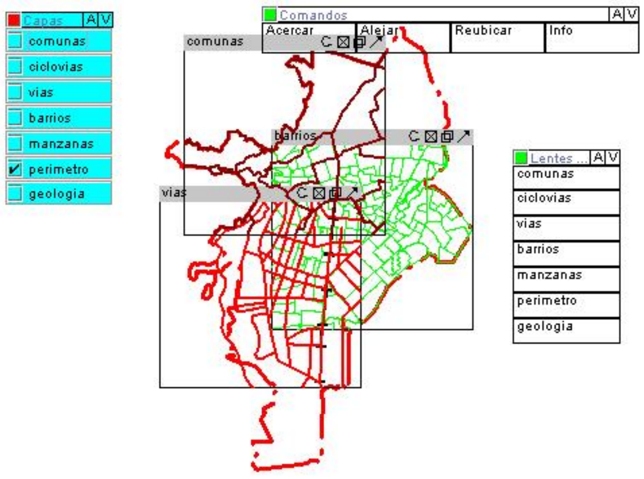

Figure 6. Features of use for the transparent API

IV. CONCLUSIONS

The Transparent API proposed is a feasible alternative to be used in graphical interfaces, such alternative allows to provide the user with a set of active tools for processing data that not interfere with the visualization.

Changing from the classical windows interface to the transparent interface at any moment is something unthinkable. The Transparent Interface is still a novel approach and there is not a commercial application that include it yet. But we can integrate some elements from these interfaces inside the real windows interfaces and thus, we can get a cohabit of the two interfaces that take advantage of the best features of each one.

In most applications, the toolbar tools occupy a region on the screen that can be best use for the application as a working area. Instead, the transparent tools do not waste this space away, since these tools are placed over the same areas but they do not prevent the visualization and interaction with the objects place underneath them because of their transparency feature.

Due to its features, the transparent tools can be used on tiny displays, such as notebook computers or personal digital assistants, cellular phones and a big variety of handheld and pocket devices that each day are becoming more common.

Also, the transparent tools can be used on big sized displays, such as wall-sized displays, where a fixed control panel might be physically out of reach from some screen positions. In this case, these tools can move with the user to stay close at hand and they do not interfere with the objects views placed underneath them.

REFERENCES

[1] BIER, A. Eric, STONE, C. Maureen, PIER, Ken, BUXTON, William and DEROSE D. Tony. (1993) Toolglass and Magic Lenses: The Transparent Interface. In: Proceedings of Siggraph '93 (Anaheim, August), Computer Graphics Annual Conference Series, ACM, p. 73-80.

[2] BIER, A. Eric, STONE, C. Maureen, FISHKIN, Ken, BUXTON, William an BAUDEL, Thomas . A. (1994). Taxonomy of Transparent Tools. In: CHI Conference Proceedings. p. 358.

[3] BURBECK, Steve. (1992). Applications Programming in Smalltalk-80: How to use Model-View-Controller (MVC). Available in: http://st/www.cs.uiuc.edu/users/march/st-docs/mvc.ht ml

[4] FOX, David (1998). Composing Magic Lenses. In: CHI Conference Proceedings. p. 25-32.

[5] GAONA, Mauricio (1998). "Magic Lens". In: "Segundo Encuentro de Ingenieria de Sistemas". ICESI.

[6] GARRETA, Luis (2001). "Arquitectura de Visualización Activa MVCL". In: Proceedings of the XXVII Conferencia Latinoamericana de Informática, p. 107.

[7] HARRISON, Beverly L., KURTENBACH, Gordon and VICENTE, Kim J. (1995). An experimental evaluation of transparent user interface tools and information content. In: Proceedings of the 8th ACM symposium on User interface and software technology, p. 81-90.

[8] HUDSON, E. Scott, RODENSTEIN, Roy and SMITH, Ian.(1997). Debugging Lenses: A New Class for Transparent Tools for User inteface Debugging. In: Proceedings of the ACM Symposium on User Interface Software and Technology, p. 179-187.

[9] STONE, C. Maureen, FISHKIN, Ken, BIER, A. Eric.(1994) The Movable Filter as a User Interface Tool. In CHI Conference Proceedings. p. 306.