19

Visual construction of highly interactive applications

O. Esteban *S. Chatty* *P. Palanque*[1]
Centre d'Études de la Navigation Aérienne
7 avenue Edouard Belin
31055 TOULOUSE CEDEX
FRANCE
{*esteban, chatty, palanque*} *@cena.dgac.fr*

Abstract

This paper presents Whizz'Ed, an experimental tool for construction of highly interactive or animated applications. Whizz'Ed makes it possible to visually describe by direct manipulation the behaviour of graphical objects and their interrelations. It provides elementary bricks that encapsulate basic interactive behaviours. Whizz'Ed aims at allowing designers to build visual programs in which the control structure is described using data-flows. The flow graph can be structured in order to reuse complex behaviours, thus allowing the designer to create new reusable bricks at design time. Whizz'Ed provide designers with a visual environment that can be used for building user interfaces as well as for database visualization.

Keywords

Visual programming, user interface, visualization, graphical interaction, direct manipulation, animation.

1 INTRODUCTION

The development of highly interactive, direct manipulation (Shneiderman, 1983) interfaces is difficult because of their inherent complexity. This complexity lies in the behaviours of the graphical objects, the relationships between them and in the dynamic management of these objects. Classic interface builders such as HP Interface Architect, TeleUSE or XFaceMaker, offer a partial solution to the problem of building graphical interfaces. Such tools make it possible to create static interfaces capable to react to user's actions. But creating more dynamic, highly interactive interfaces, such as iconic interfaces or MacDraw-like drawing tools, requires more than the instantiation and parameterization of predefined interactors. To create such highly

[1] *also with LIS, University of Toulouse 1*

S. Spaccapietra et al. (eds.), *Visual Database Systems 3*

interactive interfaces users need specialized programmers even when using high level toolkits. The challenge here consists in giving access to programming user interfaces to non-specialists in computer.science (Chang, 1987) (Williams and Rasure, 1990) by providing them with a visual programming tool for interface construction. A domain where that ease of use would be appreciated is user interface design for database systems, where ongoing research aims at providing users with simple means of representing data and interacting with them.

This paper is structured as follows. The first part describes previous work on visual systems. The second part presents Whizz'Ed, an experimental visual tool devoted to the construction of highly interactive or animated interfaces. The third part describes the implementation of our tool. We then present with a simple example showing how Whizz'Ed can be used for the design of an interactive application. The last part of the paper positions the editor according to the categorization of visual programming environments defined by Shu (1988).

2 RELATED WORK

Many studies have been carried out these last years in order to make the use of data-bases more comfortable. A new approach has been proposed based on visualization of information or data stored in traditional databases. Information are expressed with graphical representations and presented to the user in a spatial framework. Based on this approach, programming systems have been developed to offer friendly, easy to use, and understandable representations of databases. According to the literature, three main directions on visualization of databases have been exploited: the visualization of data, the visualization of data structures and the visualization of database schemas.

The visualization of data gives access to data through their graphical representations. This allows a wide range of people to manipulate these data without being trained in the use of database management systems. In order to enhance this technique, Fields and Negroponte (1977) proposed the concept of spatial data management. In spatial data management systems, the data are graphically presented in a spatial structure. Users can retrieve the information without having to know their location in the database or to learn a formal query language. They only have to go through the graphical data space using direct manipulation techniques. The View System (Friedell, Barnett and Kramlich, 1982) is such a spatial data management system. It extends the traditional systems since it generates the graphical data space dynamically according to the user's requests. SICON (Groette and Nilsson, 1988) is another tool used to navigate through a database in order to retrieve information. The information (objects) are represented by icons. Users can navigate through the set of icons. The links between objects are described by relationships such as proximity or inclusion between icons. Such systems are particularly useful for novice users with no programming knowledge.

The domain of visualization of databases schemas is still evolving. Most of progress that has been made is based on semantic database models. Hence most systems are based on semantic database model to visualize the database schemas. SNAP (Brice and Hull, 1986) is a visual editor which permits the graphical definition of the database schema. It allows the designer to navigate through the database schemas and to manipulate different entities by direct manipulation. SNAP supports the IFO model which is a formal semantic database model. SKI (King and Melville, 1984) is another visual tool for schemas browsing and/or definition. ISIS (Goldman, Goldman,

Kanellakis and Zdonik, 1985) introduces another way of visualizing database schemas. ISIS allows the user to create and manipulate entities. An entity corresponds to an object in the application environment. The new concept is the class. A class is a collection of entities of same type. ISIS allows to visualize the inheritance tree of the classes and relationships between classes as so as the attributes (data) of each class. ISIS allows users to navigate through the database at both schema and data levels. Other tools such as GUIDE (Wong and Kuo, 1982), QBD* (Angelaccio, Catarci and Santucci, 1990), PASTA-3 (Kuntz and Melchert, 1989) provide environments to visualize and manipulate database schemas in an easy way.

In the same way, the visualization of the data structures plays an essential role in giving programmers a better view of their applications. Systems for visualizing data structures provide programmers with a programming environment that enhances understanding. Kaestle (Nieper, 1983) is a graphical editor for Lisp data structures. The graphical representation of a structure is automatically generated. The graphical representations can be edited and manipulated by direct manipulation. Kaestle provides programmers with a valuable environment for the understanding of difficult structures such as circular or reentrant structures. In the same way, Brad Myers has developed a system for displaying data structures called INCENSE (Myers, 1983). INCENSE allows the programmer to design and use iconic representations in order to present data structures. INCENSE has been implemented for a Pascal-like language.

Among these different techniques of visualization, the visualization of data and data structures is close to studies on visualization of programs and program execution. These studies have been done to provide programmers with an understanding of what a program does and how it works. This step is comparable with the visualization of data-bases informations when it is used for debugging. In fact, the visualization of the variables during the execution is very similar to the visualization of the data stored in databases. In the same way, animation used to provide a dynamic description of program execution is similar to animation used to describe graphically the evolution of a data, for example in a statistical database. Systems like Balsa (Brown, 1988), Tango (Stasko, 1989) or Pastis (Müller, Winckler, Grzybek, Otte, Stoll, Equoy and Higelin, 1990) enhance visualization with adding animation. Balsa is an algorithm animation system. Pastis (Müller et al., 1990) is a program animation system based on a debugger. Tango (Stasko, 1991) is another algorithm animation system which implements a path-transition paradigm.

3 WHIZZ'ED

This section is devoted to the presentation of Whizz'Ed, a visual programming tool that allows the visual programming of highly interactive interfaces. The purpose of such a tool is to allow the creation by direct manipulation of interactive objects to build such an interface. The first part is devoted to the presentation of the conceptual model of Whizz'Ed. The second part presents the interactive editor.

3.1 The conceptual model of Whizz

The conceptual model of Whizz'Ed is based on data-flows and a set of predefined elementary components. The editor is built on top of an underlying library called Whizz (Chatty, 1992a) dedicated to the construction of highly interactive and animated interfaces. In order to explain the

data-flow model of Whizz, we usually exploit a musical metaphor. In this metaphor, graphical objects are called dancers and non-graphical objects can be either instruments or tempos. Flows between these objects are made of simple pieces of data such as integers, colors, positions, etc. Referring to the musical metaphor, these pieces of data are called notes, and are produced by instruments. In order to provide communication, each object holds several input and output slots called plugs. Dancers listen to instruments and their graphical appearance (shape and visual attributes) changes according to the notes they hear. There are different kinds of instruments, some of which will be described in the next section. Among them, for instance, are instruments called rotors, each producing positions bound on a circle, thus allowing dancers connected to them to move along that circle.

Time-based behaviours are implemented by tempos that produce notes called pulses at regular time intervals. Tempos can be used to drive instruments. For each beat of a tempo, the instruments connected to it produce a note, then received by the dancers connected to them. There is no one-to-one relation between instruments and dancer as several dancers can listen to the same instrument, and one dancer can listen to several instruments at a time. Tempos can also be shared amongst instruments. This makes it possible to describe synchronized behaviours easily. For instance, in a text editor, when a user depresses one of the arrows at the ends of a scrollbar, the cursor of the scrollbar moves while the text is scrolled. These two synchronized behaviours can be obtained by connecting the text and the cursor, which are two dancers, to two instruments that describe their movements, and that are connected to the same tempo. Finally, other sources of movement than tempos can be used. Special instruments spontaneously emit the successive positions of the mouse, or the characters typed on the keyboard. Other instruments behave as active values, and can be used to communicate between Whizz constructions and other pieces of software: they are variables, that can be modified in a program, as well as instruments, that emit their new values when modified. Flows from different sources can also be combined, for instance when using several input devices at a time: Whizz has been successfully used to implement two-handed input (Chatty, 1994).

When building highly interactive or animated systems, specifying the movement of objects is not enough. It often happens that one wants to perform an action when an object has finished its movement, or when it passes a border, or even when it meets another object. Such events are generally difficult to compute beforehand, and it is much more practical to be notified when they occur. Whizz provides a number of active zones (or fields) in order to detect such events. They range from linear borders to elliptical fields or grids. They can emit events such as crossing, entering, leaving, etc. This allows for example the easy building of a maze game. Walls are built from these kind of active zones and are automatically impassable by user's actions. Similar zones can be used for building a slider to represent the interface of a thermometer. Events may also be emitted by instruments when their part is finished, or when a particular time is reached. This enables us to use the event model that has proven to be useful for interactive applications, so as to mix interaction with animation.

The underlying data-flow model of Whizz is based on two types of information propagation: streams, used for evolutions which represent a continuous phenomenon and events, used for isolated evolutions. Events are similar to those used in many graphical toolkits and represent a less structured way of communication than streams. With this stream-event model, user's actions and animation can be mixed since the model has a unique information propagation scheme. Events may also be handed to callback functions that may reconfigure the flow graph,

by creating or destroying bricks, or changing connections. This allows events to change at run-time the behaviour of an object, or to fire animations.

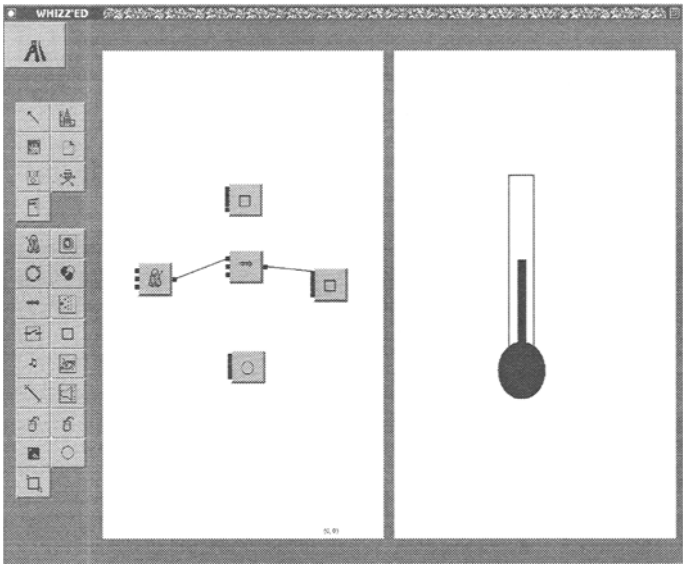

Figure 1 Animating a thermometer: the top left corner of the deformable rectangle is connected to the output of a straight trajectory. The trajectory emits positions when it receives pulses from the tempo.

Whizz'Ed provides a kit of components that can be wired together to build new components. This technique is usually called the building game metaphor. This technique is used by the Lego-Logo (Resnick, 1993) construction kit which is a rich construction environment which allows the building of creatures made with electronic bricks like sensors, motors, lights, logical-gates, flip-flops, etc. Several visual programming systems use this efficient building concept linked with a direct manipulation editor (for example see Fabrik (Ingalls, Wallace, Chow, Ludolph and Doyle, 1988)). Whizz'Ed provides a set of elementary bricks that can be dancers, instruments or tempos and makes it possible to build higher level bricks that can be seen as reusable components. When two bricks are graphically wired by the designer, these two bricks are connected and the data-flow is created.

Whizz'Ed proposes a visual language representing instruments, tempos and dancers by icons, and data-flow by lines connecting these icons. Plugs are graphically represented by small rectangles coupled with the icon representing the object. The notes are put in these plugs, and the type of the note must correspond to the type of the plug. The shapes of plugs depend on their types.

Figure 1 illustrates the conceptual model of Whizz'Ed and its user interface on a simple example of animated interface. This example describes the animation of a thermometer moving according to a predefined straight trajectory, each of its movements occurring at a given time

interval. A tempo, an instrument (a glider) and two dancers (the deformable rectangle and the fixed rectangle) are represented on figure 1. Besides the icon of the tempo are three input plugs: the first one (of type date) corresponds to the interval of time between two notes it must produce, the second one (of type integer) represents the number of notes it has to produce, and the last one (of type boolean) represents its state: active or idle. On the other side of the icon, there is one output plug of type pulse allowing the tempo to send pulses to instruments. The second icon represents the glider responsible for the straight trajectory of the rectangle. The glider has three input plugs used for the calculation of the next position of the deformable rectangle. The first input plug (called Step) is designed for receiving pulses. Each time a pulse is sent through this input plug, the glider computes the next position on the straight trajectory according to the current position. The second input plug computes a position from a number representing a linear abscissa. The third input plug called "Projection" computes position by projecting given points onto the trajectory. In all cases the computed position is sent through the output plug. The two rectangles are also represented by icons featuring five input plugs and no output plug. A dancer is always a leaf of the data-flow graph, that is to say, never featuring an output plug. From top to bottom, the three first plugs of the dancer correspond to the TopLeft, BottomRight and CenterPosition of the rectangle. The two other plugs control the BorderColor and the FillColor of the rectangle.

The data-flow is graphically represented by a link between the tempo and the glider and by another between the glider and the deformable rectangle. This small visual program behaves as follows: the tempo produces a note in its output plug that is put in the input plug of the glider. The glider reacts to this note by calculating the next position of the deformable rectangle. This position is put in its output plug that is related to the input plug of the rectangle. As soon as the rectangle receives the note (the new position for its topleft corner) it changes its position on the screen. The window on the left side of figure 1 shows the visual program corresponding to the previous animated application. This program has been built using a palette that can be used through a direct manipulation style of dialogue. The right side of figure 1 shows the execution of the program described on the left side. In this window, the straight trajectory of the glider is hidden but it can be shown when editing the program. This corresponds to the internal parameters of an object, and they can be accessed interactively by double-clicking on their icon.

This simple example could be modified easily in order to describe graphically the content of a file or a database. Instead of using a brick tempo to generate sequentially the next positions of the thermometer, these positions could be computed using values read from a database or a file by a specialized brick. If the reading is synchronized by a tempo, the thermometer will show the values of a database or a file in an animated way.

3.2 The interactive editor

Whizz'Ed consists of three main parts: the palette, the editing area and the simulation area.

- The palette contains the graphical representation of the objects (cf. left side of figure 1). This part is devoted to the elementary bricks that are supplied to the designer. The chaining of these elementary bricks allows to build complex behaviours avoiding the limitations introduced by the use of complex bricks. However it is possible for the designer to build compound bricks and to declare them as elementary bricks, thus increasing the set of bricks

primarily proposed. The palette is divided into three parts: the set of instruments, the set of dancers and the set of other bricks that are only used by designers to relate instruments or dancers.

- The editing area. This area allows to program highly interactive or animated interfaces using elementary bricks. The interface designer can use the palette (as in classical drawing tools) by direct manipulation, selecting an icon in the palette, dragging it to the editing area and dropping it at the desired place. Links between bricks are also built using direct manipulation, by selecting a plug of a brick and dragging and dropping it over another one. The arc is then automatically represented. The editor checks automatically that the type of the plugs are compatible. Bricks can be moved to simplify wiring. An example of the edition window is shown on the center of figure 1.
- The simulation window. This window aims at graphically representing the execution of the visual program built in the editing area. The simulating area guards against structural errors since the building of interactive objects is immediately visible and the appearance and behaviour of these objects is directly simulated. This window can be used either for debugging, rapid prototyping or simulating the program.

The elementary bricks are the foundation of the building of components of the interface. Some bricks are computational, while others provide user interface functions. The designer defines the application by directly manipulating the iconic representations of these bricks. The designer selects iconic representations of appropriate bricks, places them in the editing area and connects them to achieve the objects with the desired functionality and appearance. This reusable composite brick is immediately provided to the designer in the tools area. From this state, the designer can use the new brick as a classical elementary brick. We will not describe all elementary bricks that are provided by Whizz'Ed, but some of them will be detailed as they are used in an example in section 5 of this paper.

4 OBJECT-ORIENTED VISUAL PROGRAMMING SYSTEM

In this section, we present the implementation of Whizz'Ed. All the functionalities presented in the palette of the editor are fully implemented. This implementation has been done using the Whizz library (Chatty, 1992b). Whizz is based on C++ and exploits the Xtv library (Beaudouin-Lafon, Berteaud, Chatty, Baudel and Fekete, 1991) and the functionalities of the Net Audio Server (NCD, 1993) (cf figure 2). Xtv is designed for the development of direct manipulation applications. The current implementation of Xtv is on top of the X Window System from MIT. The Net Audio Server is a system developed by NCD™ for playing, recording and manipulating audio data over a network.

5 EXAMPLE: THE BUILDING OF AN ANIMATED SCROLLBAR BUTTON

We present in this section an example of the use of Whizz'Ed for building the behaviour of one of the buttons presented at each extremity of a scrollbar. The behaviour of such a button is the following: while the user points at it with the mouse and holds the left button down, the button blinks and the cursor of the scrollbar moves. The blinking of the button is obtained

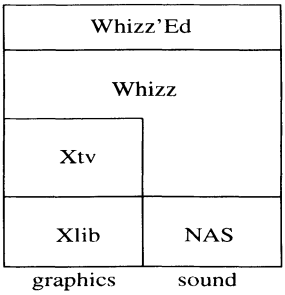

Figure 2 Implementation of Whizz'Ed.

by successively displaying it in different colors. At the beginning, the color of the button is white. When the user presses the button, it starts changing color. Moreover, in parallel with the color changing, a callback function is repeatedly called in order to move the cursor and possibly control other operations. Finally when the mouse button is released, the scrollbar button holds the current color. The behaviour of the button is relatively complicated as it combines graphical interaction such as user's actions (the click), temporal aspects (color changing) and semantic behaviour such as the callback function (for example, the text scrolling in a text processor).

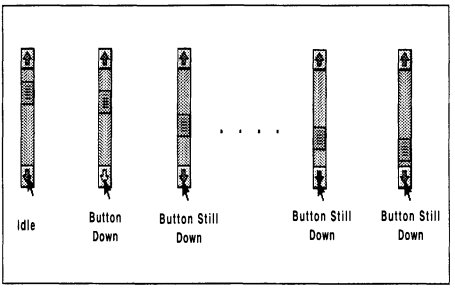

Figure 3 An example of the behaviour of the animated scrollbar button.

In spite of this relative complexity, the modelling of this behaviour is quite simple, using the conceptual model of Whizz. Figure 4 shows the behaviour of this object in terms of elementary bricks and connections. The behaviour is obtained by six elementary bricks tied together. In figure 4 only connected plugs are graphically represented as this figure corresponds to a simplified model of the behaviour of the scrollbar button done by hand. Figure 4 is organized as two high level bricks, the first one representing the detailed model of the behaviour of the animated colored scrollbar button, and the other representing the callback function related to the button. The model only describes how the connection between the two bricks is done, as the internal behaviour of the callback function is beyond the scope of this paper.

Figure 5 shows the visual program built with Whizz'Ed corresponding to the model of figure 4. First we will present the set of items (bricks and connections) of the visual program, then we will show how the behaviour of this program corresponds to the desired behaviour of the animated scrollbar button.

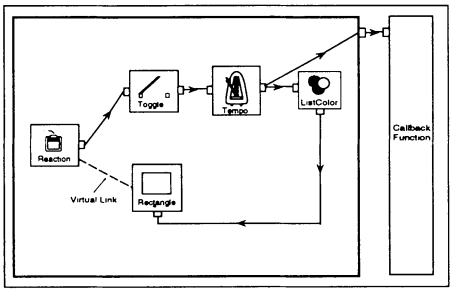

Figure 4 The informal model of the scrollbar featuring an animated button.

5.1 The items of the visual program

This section presents the set of bricks used in the construction of the visual program.

- The Reaction brick: A Reaction brick is reactive to user's actions (mouse click, mouse move, etc). It is associated with an object defined as being a sensor of the user's action and transmits this action to a connected object. These bricks are aimed at allowing user's actions to interfere with the behaviour of the system. A Reaction brick has one output plug for sending a note to the connected object when an user's action is performed on the sensor object. The designation of the sensor object is achieved by a semantic connection represented by a virtual temporary link between the reaction brick and the object which will be used as sensor.
- The Rectangle brick: Rectangles are deformable rectangles. Deformations can be achieved through the five input plugs as explained before.
- The Toggle brick: A Toggle is an integer instrument that produces alternatively ones and zeroes. It allows, for example, to start and to stop the running of another brick.
- The Tempo brick: Tempos are designed to synchronize animated actors. There is no synchronization between different tempos.
- The ListColor brick: A ListColor is a list which manages a set of colors. It features four input plugs that we will not describe in detail and one output plug. Each time it receives one event in its input plug, the ListColor emits a color note on its output plug.

5.2 The behaviour of the visual program

The visual program presented in figure 5 behaves as follows. The scrollbar button has to be reactive to mouse-down and mouse-up events. This sensitivity to user's actions is managed by the elementary Reaction brick called ButtonDn. As the button (represented by the rectangle) is directly concerned by these actions, we related the Reaction brick ButtonDn and the Rectangle brick called ScrollFrame by a virtual connection done by the drag of a invisible link between the two bricks. In order to produce the different colors that are to be integrated in the graphical representation of the scrollbar button, we add the elementary LisColor brick called ScrollColor. As the graphical representation of the scrollbar button is interested in these colors, the output plug of ScrollColor is related to the second input plug of the scrollbar button. In order to feed the input

plug of ScrollColor with pulses, we introduce the elementary Tempo brick called ScrollTempo. ScrollTempo is controlled by the elementary Toggle brick called ScrollToggle. The On/Off input plug of ScrollTempo is connected to the output plug of ScrollToggle. ScrollToggle is itself commanded by ButtonDn linked to ScrollFrame. The dotted lines in the model represent the flow of events between the ButtonDn responsible for the reaction to user's actions and the graphical object where this user's action can take place (the Rectangle brick called ScrollFrame). ScrollToggle is on (allows information to cross it) as long as the button is pressed and off when the button is released. When turning to on, ScrollToggle starts ScrollTempo. As soon as the button is released, ScrollToggle, which is now off, stops the tempo.

When the user clicks on the rectangle button, the Reaction brick ButtonDn emits on its output plug a pulse towards the Toggle brick ScrollToggle. This ScrollToggle starts the Tempo brick ScrollTempo which emits pulses as long as the button is down. These pulses activate the ListColor brick ScrollColor. ScrollColor send RGB notes from its output plug to the input color plug of the Rectangle brick ScrollFrame.

In the model presented in figure 4, the Tempo is also used to realize the repetitive call of the callback function associated with the scrollbar button as long as the button is pressed ; this relation is graphically represented by the link between the Callback brick and the Tempo brick.

This animated scrollbar button can be encapsulated in a composite brick featuring a set of input and output bricks that can be directly manipulated by the visual programmer, and can be reused in other applications as the predefined elementary bricks. Since the composite brick is totally independent of the callback function, full reusability is ensured.

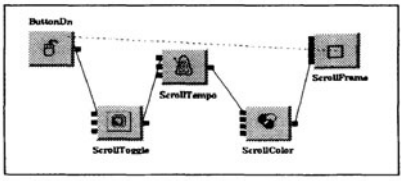

Figure 5 The visual program of the animated colored scrollbar button.

6 CLASSIFICATION INTO THE SHU CATEGORIZATION

In this section, we present how Whizz'Ed is related to the Shu (Shu, 1988) categorization. These last years, visual programming has progressed in two major directions. In one direction, graphical techniques and pointing devices are used to provide a visual environment for program construction and execution (information presentation and software understanding). In this direction, languages are designed to handle visual information, to support visual interaction and to program with visual expressions. In order to catalogue these trends, Shu proposed a categorization of the visual programming systems. She has identified two generic classes: the visual environment and the visual languages. The first class contains four subclasses:

- Visualization of data or information about data,
- Visualization of program and/or execution,
- Visualization of software design,

- Visual coaching.

The second class groups five others subclasses:

- Visual languages for handling visual information,
- Visual languages for supporting visual interactions,
- Visual languages for programming with visual expressions: Visual programming languages divided into diagrammatic systems, iconic systems and form systems.

Whizz'Ed belongs to the third subclass of the second group: the visual programming languages. It belongs more precisely to the diagrammatic systems class thanks to its data-flow model. But it can be also classed in the iconic systems class. Shu provides three criteria to assess a visual programming language. The first criterion is the level of the language. This criterion is an inverse measure of the amount of detail that a user has to give to the system in order to achieve the desired result. The second criterion is the scope of the language. This depicts the applicability of the language ranging from a general application to a specific application. The third criterion is the extent of visual expressions. This criterion is a relative measure of how much visual expressions are incorporated in the programming language.

According to these criteria, we can create the profile of the language. Whizz'Ed has a good language level thanks to its elementary bricks. The scope of Whizz'Ed is obviously restricted to the realm of user interface design. But the range of these interfaces is large from direct manipulation interfaces to animated interfaces. Moreover, Whizz'Ed uses bricks which represent visual expressions, therefore the third criterion is obviously respected.

7 CONCLUSION AND FUTURE WORK

In this paper we have presented Whizz'Ed, an experimental visual programming tool for highly interactive interfaces. Whizz'Ed allows to describe the behaviour of graphical objects in a visual way through the use of a direct manipulation editor, with a building block metaphor. A kit of elementary bricks oriented toward user graphical interaction is provided. The data-flow model ensures the dialogue between the elementary bricks, including time events and user's actions. It can be dynamically reconfigured in reaction to these events. Whizz'Ed can also be applied to databases in two main directions: the construction of visual representation of database schemas and the visual construction of interfaces to manipulate database information. In the first direction, Whizz'Ed allows designers to construct graphical representations of the schema of the database using specialized bricks. In the second direction, Whizz'Ed allows designers to visualize data using animation to represent dynamic data for example. At the moment, we are exploring directions for improving the tool. In the future, new functionalities can be added by extending the set of elementary bricks. We aim at creating a library of components to be able to define whole interfaces. For example, we need further investigations concerning the addition of sound bricks. Such an extension may be useful for multi-media interfaces. Another important next step is to create a library of functions. This library will make it possible to integrate existing domain-specific functions, by encapsulating them into bricks. We also wish to add a programming by example module to enable the design of behaviours in an easier way for end-users.

8 REFERENCES

Angelaccio, M., Catarci, T. and Santucci, G. (1990). Query by diagram*: A fully visual query system, *Journal Of Visual Languages and Computing* **1**: 255–273.

Beaudouin-Lafon, M., Berteaud, Y., Chatty, S., Baudel, T. and Fekete, J.-D. (1991). X_{TV} version 2.0 programmer's manual, *Technical report*, Laboratoire de Recherche en Informatique.

Brice, D. and Hull, R. (1986). Snap: A graphics-based schema manager, *Second IEEE International Conference on Data Engineering*.

Brown, M. H. (1988). *Algorithm Animation*, ACM Distinguished Dissertations, The MIT Press.

Chang, S. K. (1987). Visual languages: A tutorial and survey, *IEEE Software*.

Chatty, S. (1992a). Defining the behaviour of animated interfaces, *Proceedings of the IFIP WG 2.7 working conference*, North-Holland, pp. 95–109.

Chatty, S. (1992b). *La construction d'interfaces animées*, PhD thesis, Université de Paris Sud.

Chatty, S. (1994). Extending a graphical toolkit for two-handed interaction, *Proceedings of the ACM UIST*.

Fields, C. and Negroponte, N. (1977). Using new clues to find data, *Conference on Very Large DataBases*.

Friedell, M., Barnett, J. and Kramlich, D. (1982). Context-sensitive graphic presentation of information, *ACM Computer Graphics*, Vol. 16, pp. 181–188.

Goldman, K., Goldman, S., Kanellakis, P. and Zdonik, S. (1985). Isis: Interface for a semantic information system, *ACM SIGMOD International Conference on the Management of Data*.

Groette, J. and Nilsson, E. (1988). Sicon, an iconic presentation module for an E-R database, *7th ER Conference*, pp. 137–155.

Ingalls, D., Wallace, S., Chow, Y., Ludolph, F. and Doyle, K. (1988). The Fabrik programming environment, *IEEE Workshop on Visual Languages*, pp. 222–230.

King, R. and Melville, S. (1984). Ski: A semantic-knowledgeable interface, *10th VLDB Conference*, pp. 30–33.

Kuntz, M. and Melchert, R. (1989). Pasta-3's graphical query language: Direct manipulation, cooperative queries, full expressive power, *Fifteenth Conference on Very Large DataBases*.

Müller, H., Winckler, J., Grzybek, S., Otte, M., Stoll, B., Equoy, F. and Higelin, N. (1990). The program animation system PASTIS, *Technical report*, Universität Freiburg, Institut für Informatik.

Myers, B. A. (1983). Incense: A system for displaying data structures, *ACM Computer Graphics*, Vol. 17, pp. 115–125.

NCD (1993). *NCDaudio, V2.0: Overview and Programming Guide*, Network Computing Devices, Inc.

Nieper, H. (1983). Kaestle: a graphical editor for Lisp data structures, *Technical Report 347*, Computing Institute, University of Stuttgart.

Resnick, M. (1993). Behavior construction kits, *Communications of the ACM* pp. 66–71.

Shneiderman, B. (1983). Direct manipulation: a step beyond programming languages, *IEEE Computer* pp. 57–69.

Shu, N. C. (1988). *Visual Programming*, Van Nostrand Reinhold.

Stasko, J. T. (1989). *TANGO: A Framework and System for Algorithm Animation*, PhD thesis, Brown University.

Stasko, J. T. (1991). Using direct manipulation to build algorithm animations by demonstration,

Proceedings of the ACM CHI'91, Addison-Wesley, pp. 307–314.

Williams, C. S. and Rasure, J. R. (1990). A visual language for image processing, *IEEE Workshop on Visual Languages*.

Wong, H. and Kuo, I. (1982). Guide: A graphical user interface for database exploration, *Conference on Very Large DataBases*.

9 BIOGRAPHY

Olivier Esteban is a PhD student with the University of Toulouse and CENA (Centre d'Études de la Navigation Aérienne). His research interests include visual programming, human-computer interaction and construction tools for user interfaces.

Stéphane Chatty is head of the User Interface Engineering group at CENA. He is a member of the IFIP Working Group 2.7 (13.4). His current research activities include models and tools for building interactive systems, and new interaction styles, applied to air traffic control. Chatty graduated from École Polytechnique, France, and École Nationale de l'Aviation Civile, France. He received a PhD in computer science from the University of Paris-Sud.

Philippe Palanque is a lecturer at the University of Toulouse, and mainly works on formal aspects in the design of interactive systems. Palanque is a member of the IFIP Working Group 13.2. For 1995 he is co-chair of the Eurographics workshop on Design, Specification and Verification of Interactive Systems and co-editor of the book "Critical issues in User Interface Systems Engineering" to be published this year.