The dialog tool set : a new way to create the dialog component

Guilaume Texier, Laurent Guittet, Patrick Girard

LISI/ENSMA
Téléport 2 – 1 Avenue Clément Ader - BP 40109
86961 Futuroscope Chasseneuil Cedex
Téléphone: (33) 05 49 49 80 67
Télécopie : (33) 05 49 49 80 64
texier, guittet, girard@ensma.fr

1 Introduction

To help graphical and interactive applications designers in their task, several architecture models have been proposed. They all recommend to separate the application in three components which are the presentation, the dialog component and the specific component. Beyond these architectures, several tools have been proposed to create these components. These tools can be divided in two great families: the presentation toolkits called toolkits in the following and the top down generators.

The toolkits are made of different classes that represent presentation elements called widgets. A crude control mechanism is offered by toolkits. It is a matter of the callbacks which are used to link the widgets to specific component actions, regarding the events received by widgets. This control mechanism can be easily used to create a small application, but when the application enlarges the encapsulated code in the callbacks becomes hard to write and to maintain. Meanwhile, toolkits are universally used, and the method to create an application which consists of using the callbacks to define the dialog component is very spread. Moreover toolkits permit to locally and dynamically modify the application because their widgets can be dynamically instantiated and taken into account by the application.

The top down generator and particularly the model based systems (Szekely, et al., 1993) use specific languages to describe the application and are able to generate the dialog component and the presentation component from this description. These tools permit to explicitly design the dialog component of the application which is necessary to write and to maintain the application when it becomes big (many actions provided by the dialog component) and/or when the dialog is structured (a structured dialog is a dialog mode where a task can use the result of an other task to be executed). Meanwhile, there are few or no top down generator that permit to dynamically and locally modify the dialog component. With most of them, the dialog component can be modified only by changing its description and then the application have to be rebuilt.

Our proposition is to use new tools as presentation widgets which permit to create explicitly the dialog component. This dialog specific toolkit is called the **dialog tool set**. It is composed of several classes which are reification of dialog elements. Thus, any people that understand how to create the presentation component using widgets could easily define a dialog component able to manage a complex dialog like structured dialog. So, these persons would be able to easily create more complex applications.

This tool set is completely independent from the presentation component. So, its behavior is completely disconnected from the manner the user provides data. Then, it can be used to easily create multi-modal applications allowing a universal access to the application functionality.

In the following section of this paper, the concept of presentation toolkit is discussed, and particularly the way to design the presentation using this kind of tools. The next section deals with the dialog tool set, the similitude and the differences between the dialog tool set and the toolkits are pointed out.

2 The presentation toolkits

A presentation toolkit (or toolkit) is a library of interaction objects called widgets. The presentation of an interactive application is created by instantiating widgets as windows, menus, buttons, etc. The widgets have two important roles in the presentation component :

1. They are in charge of transforming information that come from input devices (keyboard, mouse, microphone,…) into application usable data.
2. They offer services to manage output devices (screen, speakers, force feedback devices, …), permitting the system to show its state.

Several advantages are offered to the designers by toolkits. First, they permit to reuse interaction tools which have a higher abstraction level than the drivers provided by the input/output devices. Then, the implementation of the presentation component is really simplified. Second, the aspect and the behavior of the widgets are the same for all the application built with one toolkit. This make the application more usable regarding the familiarity of the application (Dix, et al., 1998).

2.1 Presentation definition

The aspect definition of interactive applications is entirely devoted to the toolkit regarding the Arch (Bass, et al., 1991) architecture model. To create the presentation component, several widgets are instanced. Then, at running time, the user interactions are transformed by a specific program (often called event handler) in event representing the user action on a widget. When an event is received by a widget, it has two behavior: an internal behavior and an external behavior.

The internal behavior represent a self reaction of a widget to an event. As an example, when the user push on the mouse button and when the mouse pointer is on a button, this last one sinks. Generally, the internal behavior can not be modified. Some widgets have only an internal behavior (menu buttons are only used to display menus).

The external behavior represent the application reaction to the user interactions. This behavior has to be explicitly described by the designer with callbacks. Callbacks are functions which are called when an event is send to a widget. They are different for each widgets and for each events. Thanks to this mechanism, the application can be directly controlled by the presentation component. Indeed, specific component actions can be called using callbacks. In this case, the toolkits insure two services which are a presentation service and permit an application control role.

Meanwhile, big problems can occur when the same callback has to perform a different work regarding the application state. As an example, a mouse click may has several meanings. It can be used to select an object or to compute a position for the system. Then, to manage this ambiguity, a lot of information have to be registered in the presentation component. In consequence, the code of the presentation component highly grows up when a new functionality is added to the application. Moreover, the experience has shown that real interfaces contain often hundreds of callbacks, which makes the code harder to modify and maintain (Myers et Rosson, 1992). That's why, the architecture models for interactive systems recommend to separate the presentation and the control.

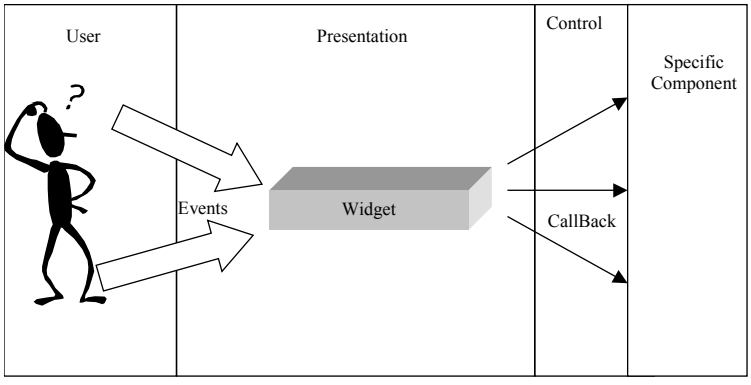

Figure 1: Interactive system control

The Figure 1 shows the manner toolkits control interactive systems using the event/widget/callback paradigm.

The way to design an application using a toolkit for presentation and control is very spread. Because widgets have several advantages.

- Firsts, they are very generic, they can be reused for any kind of applications.
- Second, they can be directly instanced using a standard programming language.
- And third, they can be dynamically instanced, so they allow the user to adapt the interface of its system to make it more usable (Dix, et al., 1998).

Meanwhile, they can not be used to control a structured and multi-object dialog. For this kind of dialog, the dialog component must be explicitly described (Pierra, 1995), and the callback mechanism can not be used because it represent an implicit dialog control that modelize only interactions but neither the application state nor the dialog state.

In the next section, the dialog tool set we have proposed is demonstrate. We focus on the ease and on the similarity of use of these tools and the presentation toolkits.

3 The dialog tool set

The dialog tool set is a set of gadgets that permit a designer to create a dialog component (respecting the ARCH (Bass, et al., 1991) architecture model and based on the element defined in H^4 (Pierra, et al., 1995) by the same way he/she makes the presentation component using widgets.

3.1 The dialog tool set classes

This tool set is composed of four family of classes that represent different elements needed to describe a dialog component supporting structured tasks and multi-objects tasks.

3.1.1 The tokens

The Tokens are the information units used to capture, at various levels of abstraction the user input (e.g. number, position, circle, and line…) in the dialogue tool set.

3.1.2 The questionnaires

The questionnaires represent the system tasks in the dialog component. In fact, they are the signatures of the functions called by the dialog component to realize treatment on the specific domain component. So, they link the dialog component with the specific domain component. They are defined by the command used by the user to activate the task they represent, by the types of tokens used by the task to call functions of the specific domain component and by the types of the returned token when the task returns a result.

3.1.3 The diagets

The diagets organize the different questionnaires of the application in abstraction levels. All the questionnaires dedicated to the creation of geometric entity are brought together in the "creation" diaget. The "Calculator" diaget is shaped by the questionnaires dealing with arithmetic's operations. Diagets contain ATN (Augmented Transition Networks) (Green, 1986) that aggregate tokens to call a questionnaire implementation.

3.1.4 The monitor

The monitor manages all the diagets. It recovers the tokens coming from the presentation and sends them to all the diagets from the lower abstraction level to the higher abstraction level.

At running time, the monitor is in charge of recovering the tokens that come from the presentation component. Then, they are transmitted to each diaget regarding the order of the hierarchy. When, a diaget received a token that it waits, it changes of state (notion of consumption), when it has received all the tokens needed to call a questionnaire, it realize this action and then it transmit the resulting token to the monitor (notion of production). Then, the monitor transmit this token to the higher diaget in the hierarchy. The fact that the diagets do not know who produces the tokens they received and who consumes the tokens they produce insure the independence between the tasks. This independence permit to define a structured dialog.

It is important to notice that the tokens representing user input are created in the toolkit events. Meanwhile the dialog component does not know the manner the tokens are created. Thus, regardless the modality used to create a token, the dialog component behavior is the same. An example of multi-modal application has been created with this

toolkit, the user could choose command with menu button or with a voice recognition system. The transition from a full WIMP application to the multi-modal application did not entail any modification of the dialog component.

3.2 Diaget vs widget

There are many similarities between the diagets and the widgets, but their respective goals are completely different. The widgets are use to create the presentation component, they are reifications of presentation concepts whereas the diaget are use to implement the dialog component, they are reifications of control concepts.

The first similarity concerns the types of elements composing these tools. To create the presentation component, two kinds of widgets are instantiated: the containers and the terminals (Fekete, 1996). The containers are only used to organize the terminal widgets in order to improve the presentation. The terminal widgets are used either to call an application functionality, either to display data, or both. By the same way, the dialog tool set has two kinds of control objects: the monitor and the diagets. The monitor is used to organize the diagets regarding the dialog semantics. The diagets are in charge of controlling the calls of the application functions.
Concerning the action calls, a similitude also exists between widgets and diagets. Widgets call actions (callbacks) when they receive some specific events. The called callback depends on the event received by the widget. Actions (the questionnaires) are called when the diaget receives a certain list of token. A similitude can be made between the widget behavior and the diaget behavior. In fact, in the dialog tool set, the diagets are like widgets, the token lists represent the events and finally, the callbacks are represented by the questionnaires (Figure 2). This similitude enlights that the proposed approach is the natural extension, for the dialog design of the way to create graphical interfaces with toolkits. Nevertheless, the proposed dialog tool set presents an essential advantage to control the application regarding standard toolkits: it permits an easy representation of the contextual aspect of user inputs because it is based on automata.

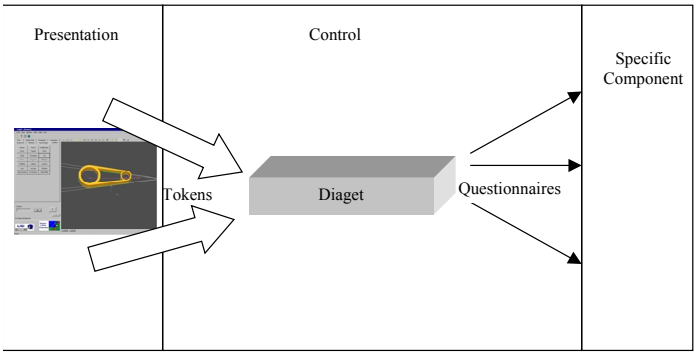

Figure 2: Diaget as widget

The last similar point between the diaget and the widget is the dynamic instanciation. Using the dialog tool set, the dialog component can be dynamically modified. Then, our tool set can be used to create an adaptive system where the user can define new functions using programming by demonstration techniques (Cypher, 1993), these functions are dynamically added to the dialog component by instantiating new questionnaires and new tokens. Anyway, this dialog tool set has been used in the TexAO (Texier et Guittet, 1999) system which is an adaptive CAD system.

4 Conclusion

Toolkits are the most used environments in the Human Computer Interaction field. They permit to reuse, rather easily, interaction tools which are friendly to create every kinds of applications. The callback mechanism appears to be adequate for the control of applications when the tasks are autonomous and slightly structured. Inversely, when the tasks are recursively decomposed into sub-tasks, the dialog language becomes strongly structured and contextual, the callback mechanism is completely inadequate. Then, the dialog component code is spread in callbacks, and the application becomes hard to maintain.

This paper presents a new kind of toolkit that permits the designer to explicitly create the dialog component. This toolkit, called the dialog toolset, is based on several classes that permit to define and to organize the system tasks in order to control their activation regarding to user inputs. The main characteristic of these tools is that they can be used exactly as the widgets. So, they keet the genericity and ease of use of the widget. The way to create the dialog

component consists of creating diagets (diaget + gadget) (representation of the widgets in the dialog toolset) and to link them to the specific domain component functions using questionnaires (representation of the callbacks) and tokens (representation of the events). Then to create the dialog component, the designer retrieves the well known method consisting of linking objects to actions via callbacks regarding events.

Moreover, the dialog toolset is perfectly adapted to the management of multi-modal application. The independence between the dialog component and the presentation component permits to adapt existing WIMP application to over modalities without modification of the dialog component.

5 Bibliography

Bass, l., Pellegrino, R., Reed, S., Sheppard, S. et Szezur, M. (1991) *The Arch Model : Seeheim revisited*. In *Proceedings of User Interface Developper's Workshop* .

Cypher, A. (1993) *Watch What I Do: Programming by Demonstration*. The MIT Press, Cambridge, Massachusetts

Dix, A., Finlay, J., Abowd, G. et Beale, R. (1998) *Human-Computer Interaction*. Prentice Hall,

Fekete, J.-D. (1996) *Un modèle multicouche pour la construction d'applications graphiques interactives*. Doctorat d'Université (PhD Thesis) : Université Paris-Sud

Green, M.W. (1986) *A Survey of three Dialogue Models*.

Myers, B.A. et Rosson, M.B. (1992)*Survey on User Interface Programming*. In *Proceedings of SIGCHI'92* (May, Monteray, CA), pp. 192-202.

Pierra, G. (1995) *Towards a taxonomy for interactive graphics systems*. In *Proceedings of Eurographics Workshop on Design, Specification, Verification of Interactive Systems* (June 7-9, Bonas), Springer-Verlag, pp. 362-370.

Pierra, G., Girard, P. et Guittet, L. (1995) *Towards precise architecture models for computer graphics: the H4 architecture*. In *Proceedings of Eurographics Workshop on Design, Specification, Verification of Interactive Systems* Bonas), .

Szekely, P., Luo, P. et Neches, R. (1993) *Beyond Interface Builders: Model-Based Interface Tools*. In *Proceedings of InterCHI93* pp. 383-390.

Texier, G. et Guittet, L. (1999) *User Defined Objects are First Class Citizen*. In *Proceedings of Third Conference on Computer-Aided Design of User Interfaces (CADUI'99)* (21-23 October, Louvain-la-Neuve, Belgique), Kluwer Academic Publishers, pp. 231-244.