DTS-Edit: an Interactive Development Environment for Structured Dialog Applications

Fabrice Depaulis, Sabrina Maiano and Guillaume Texier
LISI / ENSMA, Téléport 2, 1 Avenue Clément Ader, BP 40109, 86961 Futuroscope Cedex
{depaulis, maiano, texier}@ensma.fr
Tel : (33/0) 5.49.49.80.63 – Fax : (33/0) 5.49.49.80.64

Abstract: The structured dialog is an important concept as it's natural, for a user to break down a main goal into several sub-goals. Meanwhile, no architecture model permits the systematic design of a system taking into account such a structured dialog. In previous works, we described a new architecture model allowing this and built a library that permits a designer to create a graphical interactive application dialog controller as easily as he would build the graphical user interface. After presenting these works and comparing our approach to existing solutions, this paper introduces a graphical tool which provides an interactive development environment for designing structured dialog applications.

Key words: Model-Based Systems, User Interface, Structured Dialog

1. INTRODUCTION

In the particular field of Computer-Aided Design (or CAD), the concept of structured dialog is very important. It consists of allowing a user to decompose a main goal into several sub-goals. In a CAD system, a standard interaction sequence to create an edge between two circle centers is: Pushing the "Create Edge" button - Pushing the "Get Circle Center" button - Selecting the first circle - Pushing the "Get Circle Center" button - Selecting the second circle. Thus, the system gets the two positions it needs to create the edge.

Despite the relevance of this concept, no user interface architecture model enables such a dialog. In order to fill this gap, a new architecture

model – H^4 – was designed in our lab, several years ago [3]. It describes a dialog controller that takes the structured dialog into account. Then, we built a programming library: the Dialog Toolset (or DTS). This permits a user to create the dialog controller of an interactive application as he would design the graphical interface of an application with a standard toolkit [8]. The last step of our study was to provide a graphical tool which generates the H^4 dialog controller and verifies some useful dialog properties.

In this paper, we introduce the field that prompted us to create a new architecture model. Then, we present the existing architectural solutions and explain why they seem deficient for our study. In a third part, we outline our own model and the DTS. Finally, we show what our DTS Editor can do.

2. STRUCTURED DIALOG

Norman [7] argues that breaking down goals into sub-goals is the natural way for users to solve a complex problem. This decomposition is done recursively until the actions correspond to system functions. Such an analysis highlights the "structured tasks" paradigm that is a very important concern in the application field we address, i.e. the CAD.

During a structured dialog, each task can be executed in the context of another task. It is able to provide results that can be used by a higher abstraction level task. On the one hand, a hierarchical task organization results from this feature: **production** tasks (e.g. getting a circle center) are lower abstraction level tasks than **terminal** ones (e.g. edge creation). On the other hand, different tasks can be gathered in the same abstraction level. Thus, three different levels stand out : the "creation" level concerns terminal tasks; the "compute" level concerns tasks that extract object features; the "select" level concerns tasks used to select entities.

In the next part, we will study the different solutions used to create interfaces, and show why they are unable to integrate such a dialog.

3. EXISTING SOLUTIONS

In order to create interactive graphical interfaces, [1] suggests two main ways: the bottom-up and the top-down methods.

The bottom-up method consists in creating the presentation layer, using a toolkit. A toolkit is composed of widgets. It permits a very simple control mechanism, using callback functions, that are activated as soon as any event occurs on a widget. This corresponds to the application behavior. To create an application with this method, the designer realizes the functional core.

Then he creates the presentation layer and, finally, he links the presentation to the functional core with the callback functions. The main issue of this approach is that the dialog logic is spread through all the application widgets. Indeed, each one has a predefined behavior that represents a part of this dialog. In [5], E. Lecolinet presents a new way for designing interface with an original toolkit : UBIT (Ubiquitous Brick Interaction Toolkit). It manages to make the interaction control easier and to increase the designer freedom. Meanwhile the obtained behaviors cannot substitute for a genuine dialog controller.

The top-down method enables a designer to describe the application interface with a specific language. This description is used to generate the codes of the dialog controller and of the presentation. In this context, designing applications consists in : creating the functional core, describing the interface and, finally, linking the generated code to the functional core. In contrast to the generic bottom-up method, the top-down approach based systems (or model-based systems) are specialized in a peculiar application type. As an example, a generator of drawing applications is not able to build word processing software. Moreover, only experimental systems have been made with this method. The dialog of an interactive application can also be described in a formal way, e.g. with high level Petri nets. This approach permits, via a Petri nets editor, to specify the application dialog dynamically [6]. This method enables a task structuring: a Petri net that represents a given task is able to call an other Petri net. But, as a Petri net high abstraction level task contains a lower abstraction one, the dialog logic must be planned.

As a conclusion, it seems that no solution permits to create a dialog controller that enables both structured dialogs and independent calls of tasks.

4. THE DIALOG TOOLSET

The architecture model H^4 has been created for CAD system designers. It is based on the Arch model [2] and details the role and the structure of any component. It involves a dialog controller, which manages structured dialogs, and consists of several elements described below.

4.1 Dialog Toolset Components

Tokens are information units that connect tasks together : the *"command tokens"* permit tasks activation, while *"parameter tokens"* hold typed values.

Questionnaires are the task abstractions within the dialog controller. They are the function signatures that are called by the dialog controller to

modify the state of the functional core. As entries, they own one parameterized token list. Their output is a single token that carries the result of the task execution. Moreover, each one has a link to a functional core method, called when the questionnaire has received all the needed tokens.

By analogy with the widgets (window + gadget), the name of the controller interactors is **diaget** (dialog + gadget) [8]. A single Diaget gathers several questionnaires of equal abstraction level. Each one contains an Augmented Transition Network (ATN) [9] that remembers and organizes the tokens it receives. The *Figure 1* shows an ATN representing a Circle Creation Diaget : it accepts a command, a position (center) and a numeric value (radius).

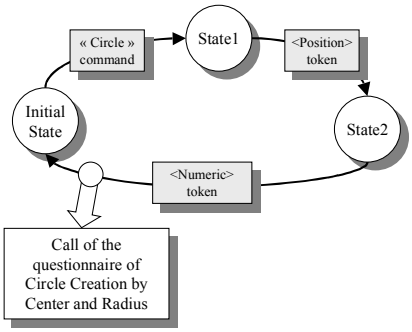

Figure 1. Dialog interactor functioning

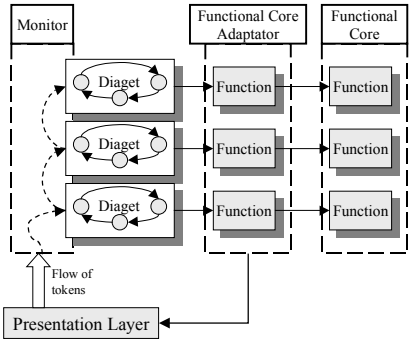

Figure 2. Dialog controller functioning

The **monitor** organizes the diagets in a hierarchical way. They are sorted from bottom to top and the rank is given to the monitor. When getting a token from the presentation layer, the monitor transmits it to the first diaget in its hierarchy. If it is not consumed, the diaget gives it back to the monitor which proposes it to the next diaget and so on (*Figure 2*). As a result, the tasks are independent, as they never know the source of the tokens they get and the destination of those they produce. Moreover, the monitor knows what tokens are expected by each diaget, and it is able to authorise only valid tokens to be produced, according to the current dialog state

From this architectural model, we have created a library that permits to build an application dialog controller like an interface is made with a toolkit: the DTS (Dialog ToolSet).

4.2 Dialog controller coding issues

Nevertheless, programming such a dialog is tedious and reluctant though, the dialog controller code is very redundant. Moreover, several dialog properties, that deal with task completness, could be automatically checked:

- **Production / consumption consistency**. Is a diaget able to receive a given type of token ? A least one lower diaget, in the monitor hierarchy, must be able to provide it. If there is not such a diaget, the corresponding system task will be available but will never be completed.
- **Questionnaires and command tokens**. Do two different questionnaires in two different diagets have the same signature ? Such a situation will prevent the higher one, in the monitor hierarchy, and consequently the task, to be executed.
- **Overloading**. Do two questionnaires in the same diaget have the same beginning of entry token sequence? In this case, one of the corresponding task won't be able to be used.

In the next section, we present the Diaget Editor we have added to the DTS, that helps developers in coding their dialog and solves this problem.

5. A DIAGET EDITOR

The Diaget Editor main goals are to provide to an interactive graphical application a development environment for creating the dialog controller code and to verify some useful dialog properties.

5.1 Editing diagets

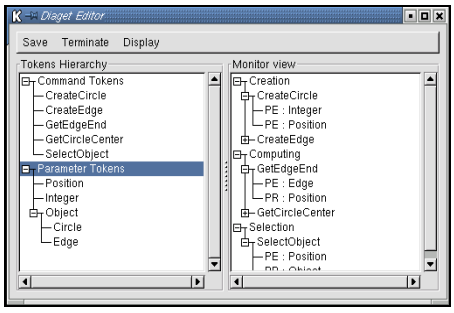

Figure 3. The Diaget Editor views

The left part of the Diaget Editor (*Figure 3*) displays information about tokens, while the right part deals with the diagets and questionnaire signatures. Direct manipulation has been largely used, and drag and drop from one side to another is possible. Saving and restoring DTS architecture is possible, either in Diaget Editor native format or in generated code format.

5.1.1 Tokens

The left part of the *Figure 3* presents the token hierarchy, which is split into two classifications. On the one hand, the command tokens carry the questionnaire name and are the only tokens able to make a questionnaire

waiting for its sequence of parameter tokens. In most applications, they are provided by the user to the Dialog Controller via interface buttons or menus. On the other hand, parameter tokens carry functional core objects (circles, edges, …). They are organized hierarchically, as the functional core objects are. Hence, some of the questionnaires can be gathered together. For example, in a MacDraw like application, the user is allowed to create circles and edges, that inherit both from a "graphical object" class. Defining a single selection questionnaire for a "graphical object" is enough and combines both circle and edge selections.

In DTS-Edit, the two token classifications are displayed. The command tokens are ranked in a single depth tree. These commands are created automatically each time a new questionnaire is defined.

Parameter tokens are illustrated by a tree, which represents parent connections. Every sub-tree inherits from the tree it is part of. These tokens are not automatically created. Defining a new parameter requires the user to give its name and to declare its parent. It is then added to the hierarchy. It is important to notice that, to be able to add a parameter to a questionnaire signature, the parameter token has first to be described in the token hierarchy.

5.1.2 Monitor

The right side of the DTS-Edit interface displays the monitor view. On this panel, diagets and questionnaires can be created, and the monitor hierarchy can be described.

Creating some diaget implies clicking on the panel and giving a string for its name. All diagets are displayed according to the hierarchy. Changing a diaget position with drag and drop operation affects the monitor hierarchy.

In order to add some questionnaire to an existing diaget, the designer just clicks on the wanted diaget and gives a name. To achieve the signature definition, he can add either entry parameters or a result (as shown in *Figure 4*).

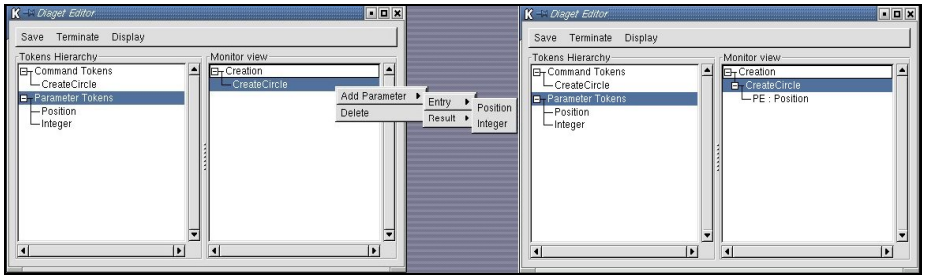

Figure 4. Adding a Parameter to a Questionnaire

Once all diagets are described, including the questionnaires and the tokens they manage, the designer can achieve the process. He clicks on the

"Terminate" button and, then, the dialog controller code is generated. Moreover, the functional core adapting function headers are created thanks to the questionnaire signatures. Then, the user is given the possibility of programming the functional core adapting function bodies via an editing window. At this time, the system checks some properties of the dialog.

5.2 Checking properties

Thanks to the architectural model, H^4, the tool is able to check several dialog properties.

- **Production / Consumption consistency**. In H^4, the diagets are sorted from top to bottom : the highest ones receive tokens from the lowest ones. According to the diaget ranking and to the tokens they produce and consume, the system knows and warns the user when a questionnaire of the highest diaget produce a not empty token and when it is impossible for a Questionnaire to receive a token that is necessary for its execution.
- **Questionnaires and Command tokens.** A questionnaire is able to receive its sequence of tokens once it has been provided with its command. This latter is usually provided by the presentation layer, via a button. If two questionnaires, set in two different diagets, have the same name, one of them will never be activated. The command token will always be caught by the lowest of the two diagets. The tool warns the user when such a situation occurs.
- **Overloading.** In the same diaget, two questionnaires can have the same name : they are activated with the same command token. Meanwhile, a problem happens if the entry list of the tokens from one questionnaire is a sub-set of the entry list of the tokens of the other one. It is impossible to guess what to do : if the diaget activates the first questionnaire, the second one will never be provided with a matched sequence of tokens. If it waits for other tokens, trying to complete the second questionnaire sequence of tokens, the first one is forgotten and will never be activated.

6. CONCLUSION – FUTURE WORKS

The existing interface architecture models do not easily enable the structured dialogs. For this reason, we have designed our own architecture model, H^4. We then wrote a library, the DTS, that permits any user to build the H^4 dialog controller like an application user interface with a toolkit.

Unfortunately, the DTS is very intricate to use, and the dialog controller code is very repetitive and error-prone. For this reason, we have created a

tool that permits to interactively build the dialog logic of a graphical application. The redundant code is generated and the coding errors are reduced. Moreover, this tool checks some useful dialog properties. Automating these tasks, DTS-Edit makes our dialog library easier to use.

The described method to produce and verify the dialog controller will be soon improved with new features. Further challenges include :

- Adding some validation concepts. In [4], the authors show how it is possible, with H^4, to check the interaction reachability and completeness. We will add this method to our tool.
- The H^4 dialog controller uses tokens, which carry some functional core information. Instead of manually creating them, we want our tool to introspect the functional core classes and create the tokens on its own.
- Generating the interface description file, that links the dialog controller to the presentation layer, from the dialog description given by H^4.

7. REFERENCES

1. Bass, L. et Coutaz, J. *Developing Software for the User Interface*. Addison-Wesley, 1991.

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

3. Girard, P., Guittet, L. et Pierra, G. H4 : un modèle d'architecture d'applications graphiques interactives. In *Proceedings of AGI'96* (6-7 juin, Tours, France), 1996, pp. 245-250.

4. Jambon, F., Girard, P. et Boisdron, Y. Dialogue Validation from Task Analysis. In *Proceedings of Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'99)* (2-4 June, Universidade do Minho, Braga, Portugal), Springer-Verlag, 1999, pp. 205-224.

5. Lecolinet, E. Réification et réplication dans les interfaces graphiques : le toolkit Ubit. In *Proceedings of IHM'99* (23-26 Novembre, Montpellier), 1999, pp. 9-12.

6. Navarre, D., Palanque, P., Bastide, R. et Sy, O. Le Prototypage aussi peut être formel. In *Proceedings of ERGO-IHM 2000* (3-6 Octobre, Biarritz), 2000, pp. 153-160.

7. Norman, D. *User Centered System Design*. Lawrence Erlbaum Associates, 1986.

8. Texier, G. et Guittet, L. Dialogue+Gadget=Diaget. In *Proceedings of Onzièmes journées sur l'ingénierie de l'Interaction Homme-Machine* (22-26 Novembre, Montpellier France), Cépadues-Editions, 1999, pp. 70-77.

9. Woods, W. Transition Network Grammars for Natural Language Analysis. *Communications of the ACM*. 13, 10 (1970), pp. 591-606.