Patry G. & Girard P. GIPSE: a Model-Based System for CAD. Third Conference on Computer-Aided Design of User Interfaces (CADUI'99), Louvain-la-Neuve, Belgique, 21-23 October 1999. p. 61-72.

Chapter 6

GIPSE, a Model Based System for CAD Software

GUILLAUME PATRY AND PATRICK GIRARD
LISI / ENSMA, Téléport 2, 1 Avenue Clément Ader
BP 40109, 86961 Futuroscope Chasseneuil Cedex
{patry, girard}@ensma.fr
Tel : (33/0) 5.49.49.80.70 - Fax : (33/0) 5.49.49.80.64

Key words: Model-Based Systems, CAD, User Interface, CADUI

Abstract: We describe in this paper GIPSE, a model based system that is specialised in CAD systems. GIPSE facilitates the design and the development of applications using structured dialogues. From a functional core and some initialisation files describing the user interface properties such as the tasks, their structuration and the presentation, GIPSE allows the online construction of self-running applications.

1. INTRODUCTION

User interface and dialogue control are a large part of any software. They are hard to implement, to debug and to maintain [11]. The problem grows much worse when the application dialogue extends, or when it follows some dialogue strategy where relations between user tasks are not well defined. Such is the case in CAD systems. The user must be able to express a lot of geometrical relations between entities, leading to a structured dialogue where parameters of tasks may be given by the way of a (possible huge) number of subtasks.

A lot of tools have been created in order to facilitate designing user interfaces. These tools range from toolkits and application frameworks to model based systems. In the latter ones, the desired user interface is automatically generated from a specification made of declarative models. However, few of these tools address the problem of structured dialogues.

Our goal in this paper is to describe GIPSE, a model based system specialised in applications where this dialogue form is usual. In the subsequent section, we explain more precisely the particularities of our main domain, CAD software. A natural decomposition of structured dialogue in a general task structure follows. Then we describe GIPSE, our specialised model based system, and give some example of its use.

2. DOMAIN

Technical design is a domain where construction constraints are largely used. In order to be helpful for end users, Computer Aided Design (CAD) software have to facilitate the expression of these constraints. Graphical applications do not usually ask the user for carefully positioning objects. On the contrary, CAD systems require the user to strictly define the geometrical entities that make up technical objects. So, systems offer a wide range of methods for constructing these entities. A software from the former type may allow the construction of an ellipse with two mouse interactions, and then may allow its modification by the way of direct manipulation. On the contrary, CAD systems allow the creation of these entities in terms of centre, radius, axis, or allow the use of constraints such as tangency or coincidence. Moreover, these parameters can themselves be defined by the way of numerical or grapho-numerical expressions. Circle creation, for example, can use a subtask that returns the projection of a position on a line to define its centre, and then use another subtask to calculate the distance between two entities as the radius. The dialogue associated with a task is recursively including dialogues associated with subtasks allowing the realisation of intermediate goals.

This kind of dialogue is called a structured one [15]. On the one hand, it gives to the user a very powerful way of expressing constructs constraints. On the other hand, using these dialogues requires specific software architecture, to reduce the combinatorial explosion of subtask calls. Some examples of these architectures may be found [4, 10, 18].

Nevertheless definition and implementation of these dialogues is very difficult. Current user interface development systems lack the ability to describe them. These systems, such as JANUS [1], MASTERMIND [20] TRIDENT [2] or MECANO/Mobi-D [16, 17], may be used in fields where the user’s tasks are well defined. Examples of such fields are database manipulation (JANUS), or stock management (Mobi-D). In such systems, the number of tasks allowed to the user at any point of the session is limited.

In contrast, structured dialogues give the user many ways of expressing his/her needs: any (sub)task able to produce a parameter from a currently waited kind must be available.

3. STRUCTURED DIALOGUES

Structured dialogues may be seen as a set of producers and consumers. Some tasks consume data produced by some other tasks. In our previous example, the "Create Circle" task gets data from two others tasks, "Projection" and "Distance", which produce respectively one position and one real value.

Tasks may be decomposed in three categories: terminal tasks, production tasks and opportunistic tasks.

– Terminal tasks use data, but do not produce[1] them. They are used to modify the application (mainly geometrical) model. Some examples are tasks allowing entity creation, destruction or modification. These tasks are the main reason for the existence of CAD systems. At any moment, only one terminal task may be active. One can not create a circle while creating a line. Two dialogue strategies may apply: either the expression of a new terminal task is forbidden (by disabling menu components), either this expression replaces the current one, which is then cancelled, with all its subtasks.
– Production tasks consume data in order to produce another data that will be transmitted to another tasks. These subtasks exist solely in the context of another task, either a terminal task or a production (sub)task. If no terminal task is in use (i.e. the user is not currently modifying the model), then no production task must be available, as there is no one to transmit data. It is the existence of such tasks that define structured dialogue. Production tasks are defined by the result they give, and not by their client (to which task they give it). The "Centre" task gives a position, and may be used to define the extremity of a line or the centre of an ellipse. Last, production tasks do not modify the application model. They may extract some information from their parameters ("Centre"), or they may calculate some value ("Distance"), but they do not create, modify or destroy any entity while doing so.
– Opportunist tasks are the third kind of tasks in structured dialogues. These are the tasks that neither modify the model, neither produce

[1] Be careful that we are interested here in dialogue. So, when we claim that terminal tasks do not produce anything, it is in the sense of dialogue: they do not produce any data (dialogue token) for other tasks. Obviously, they produce entities in the CAD system model.

information for other tasks. These tasks allow end users for asking for a modification of the system to allow better realisation of tasks. An example of this kind of task is the translation of the visible content of a window. Opportunist tasks modify the presentation of data, but do not change them. These tasks may be used at any moment, without cancelling the currently active terminal task. The former temporarily replaces the latter, until it is completed. The terminal task then resumes as if no interruption had ever been done. In our previous example of a circle construction, the user may, at any moment, ask for a zoom, define the position that delimits the zoomed area, and then resume his/her construction.

4. GIPSE: A MODEL-BASED SYSTEM

GIPSE is a model-based system specialised in software that use structured dialogues. Its main goal is to provide a basis than can be extended later to provide additional capacities. One of the stated goals was to allow for fast modification of the dialogue.

GIPSE does not generate code. It is composed of a software component that links itself with the application in order to manage the dialogue. It acts as the dialogue controller for the whole application. As the application starts-up, specifications of the dialogue properties and functional core functionalities are loaded from textual models. From this point, GIPSE manages the dialogue until the end of the session with the application.

This structure has several benefits. First, as the model is directly used by the system, there is no recompilation time whenever there is a change in the dialogue structure. Second, as the dialogue is itself a data structure, in opposition to hard coded dialogue, it is possible to dynamically modify it. The user may be allowed to remove unneeded functionnalities, or to add new ones, created by the way of Programming by Demonstration (PbD) [3]. This leads to the possible extension and specialisation of a system by the end user himself/herself [14]. Third, it is possible to integrate verification tools in the software itself. Some works have already been done on that subject [9]. This ability to examine the software dialogue, even during run time, comes handy when the end user is able to modify the dialogue. Inconsistency or incorrect dialogue structures may be notified.

In this paper, we do not focus on the PbD features of GIPSE, which is a purely interactive tool that allows developing interactive applications by extension/specialisation [6, 13, 14]. We only deal with declarative models that are used by GIPSE to manage the interactive system

4.1 Task structuration in GIPSE

GIPSE uses an architectural model called H^4 [7], which was designed specifically for CAD systems. In the H^4 model, the dialogue controller is composed of a hierarchical set of agents, named dialogue **interactors**, relatively to the theory of Interactors ([8, 12]). Each of them corresponds to a task level: they group and structure system **tasks** in a hierarchical fashion. They are defined by the nature of information they accept as input, and produce output information, which is the result of the task. Usual dialogue interactors in drawing applications include *designation* (picking graphical entities), *information* (giving information such as centre or extremity of entities), *calculus* (projection, distance), and *creation*. Communications between interactors are automatically managed by a **Monitor**. Interactors may accept information from any other interactor that is lower in the hierarchy, without knowing who produced it. The communication unit is a typed element, named **Token**, which may either be a command (order to initiate an action) or data (parameter of actions). The monitor transmits tokens to the next interactor in the hierarchy that waits for such typed data.

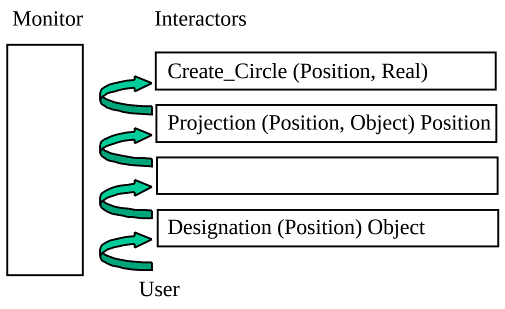
Monitor
Interactors
Create_Circle (Position, Real)
Projection (Position, Object) Position
Designation (Position) Object
User

Figure 1. Dialogue controller in GIPSE

This model of dialogue controller allows for a great deal of independence between tasks. The monitor manages communication between separate task levels. Possible interactions between tasks of the same level are managed by the interactor that manages this level. Each task can be defined independently from each other. The same thing can be said about interactors. Each one is independent from the other, and manages the tasks at its own level. The number of other interactors able to provide or consume data does not matter. An interactor provides data to and consumes data from the monitor, and transmits theses data to any active task via the monitor. One consequence of that structure is no need for complex task and dialogue models (see section 4.2).

It should be noticed that this model is focusing on dialogue control. Presentation of model objects is out of its scope. CAD entities are highly structured and relational. Their presentation on the screen is therefore left to the functional core, using some specialised software component.

4.2 Specification of CAD software with GIPSE

A system, in the H^4 model, can be specified from three components: tokens (communication unit between tasks), tasks (communication unit between user and system) and interactors (task structuring unit). The monitor can be derived from the interactors order.

The description of these components, along with a description of the presentation, allows for the complete specification of the application dialogue. The token specification is akin to the object model, whereas tasks specification constitutes the task model, interactors specification represents the dialogue model and presentation specification states for the presentation model.

4.2.1 Object model

Tokens are objects that communicate user's intention. They may contain data (references to model objects, or values), or commands initiating tasks. Commands, and some data such as string or numerical value, exist in any application. Most of them, however, are application-dependent and refer to model objects.

Tokens are a communication unit, allowing tasks to freely exchange data. The exact structure and value of these objects are of no importance to other dialogue controller objects. The interactor that manages a given task does not care what is the value of a position. What is important is the nature of the transmitted data, as it allows it to know when the task may be called (i.e. have received all parameters).

```
-- Tokens
Rectangle
Line
Point
Arc
Unbounded_Line
Circle
-- Groups
Support_Object   {Unbounded_Line, Circle}
Drawing_Object   {Rectangle, Line, Point, Arc}
Graphical_Object {Support_Object, Drawing_Object}
```

Figure 2. Object model example

The object model of the dialogue controller contains the list of application-dependent token types, and provides for a structuring mechanism

(Figure 2.). This structuring mechanism (groups) allows for concision in task description, as we will see in the next section.

4.2.2 Task model

Thanks for the H^4 architecture, tasks may be defined individually, without any connection between them. A name, a list of input parameters and one (optional) output parameter define each task. This specification allows the system to automatically disable any task that cannot give any parameter of the waited kind. As it has been said, it defines the tasks in terms of what it takes and what it produces.

Some help line may also be added to the description. This is used by the system to provide the user for dynamic help. Figure 3 shows an example of task description. In the second case, we can see the use of structuring: only one task description is required for all the graphical entities (every entities in our small example).

```
-- Terminal task
Task{
  Create_Circle (Position, Real)
  Help {Circle creation by center and radius}
  Parameter {Position, “Center”}
  parameter {Real, “Radius”}
}
-- Production task
Task {
  Center {Graphical_Object} Position
  Help {give an Object’s Center}
  Parameter {Graphical_Object, “Give an object”}
}
```

Figure 3. Task model example

Other characteristics of the task may be added for PbD purpose (see later). Such characteristics are, for example, the reaction of the task when macro-recording is on. Some tasks may be ignored (that is, not recorded), while others are strictly forbidden ("save" for example).

4.2.3 Interactor Model

The Interactor model defines the dialogue. Each interactor defines the dialogue at its own level, while the hierarchy of interactors gives an implicit rule on which task may use which other tasks. A task from a given interactor can use any task from a lower level interactor, given that, this lower task is able to produce data that are waited by the higher one.

Interactors define the dialogue at their level using a MAD-like approach [19]. Tasks are structured with Alternative (ALT) and Sequence (SEQ), possibly repetitive (ALT* and SEQ*).

For example, the “Creation” interactor definition may look like this:

```
Interacteur Creation {
  Seq {
   Alt {
     Task {New}
     Task {Open}
   }
   Alt {
     Alt* {
      Task {Create_Circle    (Position, Real)}
      Task {Create_Rectangle (Position, Position)}
      Task {Create_Line   (Position, Position)}
     }
     Task {Close}
     Task {Quit}
   }
  }
}
```

Figure 4. Dialogue model example

The end user must begin by either creating a new document or opening an existing one. It then can either create an indefinite amount of objects (the Alt*), or finish the session by closing the document or exiting the application. Other interactors are defined in the same way. Most of them are composed of a simple alternative between multiple tasks. The language constructs are very few; the communications between interactors and the behaviour of the monitor (described in 4.1) extend automatically the possibilities of the language.

This description defines the static constraints on dialogue: what is eventually available at any given moment. Dynamic constraints are managed by the system at run-time: in the set of available tasks, only tasks that produce expected data are effectively available. Others are disabled (greyed out) by the system. This is completely automatic.

The description of the dialogue in independent chunks, themselves composed of structured tasks, and linked by explicit rules, allows for an easy description of the whole dialogue. Adding a new task to the system implies only the modification of the interactor that will manage it, to have this task fully integrated with the whole application dialogue. The same description, realised via a state-transition diagram, would be unmanageable, as the number of possible transitions between tasks is exponential. The adjunction of a new task would, in this case, imply the creation of one transition from any task able to consume the produced data, and to any task able to produce waited data.

4.2.4 Presentation Model

Presentation of a CAD system may be boiled down to one (or more) visualisation space and some way to express commands and data to the system. The former does not concern us, as it is under the control of the model itself. The latter are composed of menus and dialogue boxes that allow entering specific data, such as strings, or numerical values.

The presentation model we use is based on tasks, and concentrates on ways of initiating them. We use menus and submenus as containers for labels or pictures that, when used by user, activate the corresponding tasks.

```
Menu Objet {
  Label Line   {Create_Line (Position, Position)}
  Label Circle    {
     Create_Circle (Position, Reel)
     Create_Circle (Position, Position)}
  Label Rectangle {Create_Rectangle (Position, Position)}
  Menu Modification {
     Label Color {Modify_Color (Object)}
     Label Style {Modify_Style (Object)}
  }
}
```

Figure 5. Presentation model example

As shown by the "Circle" label (Figure 5), a label may be associated with more than one task. The only restriction is that these tasks should be in the same interactor. When the user selects this label, both tasks are initiated. The system then allows unrestricted use of any task that give a parameter for any of these two tasks. A parameter is transmitted to any task that waits for it. A parameter that is accepted by one task and not by the other automatically cancels the second one. For example if the user clicks on "Circle" (initiating both tasks) and then gives a numerical value, then the task he/she is asking for is the one with the numerical parameter. The other one is then silently cancelled.

4.2.5 Example of use

We will present in this section an example of system modification. We suppose that a new functionality, defined by (and associated to) its specification "Intersection (Object, Object) Position" has been added to the functional core.

From this point, we add to the textual task model the information associated with this task.

```
…
Task {
  Intersection (Graphical_Object, Graphical_Object)
                                             Position
  Help {return the nearest intersection of the two object }
  Parameter {Graphical_Object, “ First Object ”}
  Parameter {Graphical_Object, “ Second Object ”}
}
…
```

We then add the task to an interactor. As it is a production task (it returns a "position"), that calculates its response from the parameters (in opposition to extracting information of these parameters), we place it in the "Calculus" interactor.

```
…
Interacteur Calculus {
 Alt {
   Task {Centre   (Graphical_Object) Position }
   Task {Distance (Position, Position) Real}
   Task {Intersection (Graphical_Object, Graphical_Object)
Position}
 }
}
…
```

and we add a label in a menu to allow the end user for using the task.

```
Menu Calculus {
  Label Centre   {
     Create_Line (Position, Position)}
  label Distance {
     Create_Circle (Position, Reel)
  label Intersection {
     Intersection (Graphical_Object, Graphical_Object)
Position }
}
```

From this point the new task is fully usable in the system. The user may use it as a subtask for any higher task asking for a position. In the same way, any task producing a Graphical_Object and belonging to a lower interactor may be used to give its parameters.

We describe here a purely textual way to accomplish this modification. As we stated upper, it is possible with GIPSE to do the same modification in a purely interactive way, using PbD possibilities of GIPSE. The only thing we cannot do interactively is modifying the object model, which is a work that has been conducted in our laboratory [21].

5. RELATED WORK

GIPSE is a specialised model based system (MBS). It is specialised in an application domain that presents some very specific characteristics, which influence the required dialogue structure. The main difference between GIPSE and other existing MBS lies therefore in its dialogue management system. Most MBS, as we have already stated, have few or no support for structured dialogues. They are adapted for the conception of forms applications, but are unusable for the design and conception of CAD systems.

GIPSE has some common points with SACADO [5, 10], another development environment oriented towards CAD systems. Both use the notion of hierarchical level of dialogue tasks. However, the model used and the assertion made by the system on the application are different. SACADO makes it necessary for the developer to define the set of relations between tasks: any function that may be used while creating a circle to produce a parameter must be explicitly given. On the contrary, the hierarchical decomposition described in the dialogue model of GIPSE gives an implicit set, whose exact content has not to be defined by the developer.

6. CONCLUSION AND FURTHER WORKS

As a model based system specialised in CAD systems, GIPSE facilitates the conception and development of applications using structured dialogues. Using a functional core and some initialisation files that describe the dialogue properties (the tasks, their structuring and the presentation), GIPSE allows the online generation of dialogue and the execution of the application.

GIPSE is a very flexible system, which allows the modification of the dialogue without any recompilation, by the modification of human-readable text files. Current works aim to extend this flexibility in several ways. End-user programming, that allows adding and removing user-produced tasks, has already been integrated in the system. The creation of new category of objects and associated tasks is actually in study. Another currently studied extension is the ability to mix structured dialogue and direct manipulation, allowing the user to choose his/her interaction depending on his/her goals [13]. Last, we are now integrating validation and verification tools in the system, to warn or prevent the user for creating non functional dialogues [9]. These additions will allow GIPSE to become a full CAD software development environment, an actual CADUI product.

7. REFERENCES

1. Balzert, H. From OOA to GUI : The JANUS-System, in *Proc. IFIP TC13 Human-Computer Interaction (INTERACT'95)* (Lillehammer, Norway, 27-29 June, 1995), Chapmann & Hall, pp. 319-324.
2. Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., B. Sacré and Vanderdonckt, J. Towards a systematic building of software Architectures : the Trident Methodological Guide., in *Proc. Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'95)* (Bonas, France, 1995), Springer-Verlag/Wien, pp. 262-278.
3. Cypher, A. Watch What I Do: Programming by Demonstration. The MIT Press, 1993. 604 p.
4. Fekete, J.-D. *Un modèle multicouche pour la construction d'applications graphiques interactives*. PhD Université Paris-Sud, Orsay, 1996, 203 p.

5. Gardan, Y., Jung, J.-P. and Martin, B. An End-User oriented approach to design man-machine interface for CAD/CAM, in *Proc. IEEE Internationel Conference on Systems, Man and Cybernetics* (Le Touquet, France, 17-20 Octobre 1993, 1993), pp. 525-530.
6. Girard, P., Patry, G., Pierra, G. and Potier, J.-C. Deux exemples d'utilisation de la Programmation par Démonstration en Conception Assistée par Ordinateur. *Revue Internationale de CFAO et d'informatique graphique*. 12, 1-2 (1997), pp. 169-188.
7. Guittet, L. *Contribution à l'Ingénierie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO*. PhD Université de Poitiers, 1995, 196 p.
8. Harrison, M.D. and Duce, D.A. *A review of formalisms for describing interactive behaviour*. University of York, January 7 1994.
9. Jambon, F., Girard, P. and Boisdron, Y. Dialogue Validation from Task Analysis, in *Proc. Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'99)* (Universidade do Minho, Braga, Portugal, 2-4 June, 1999), Conference proceedings, pp. 201-221.
10. Martin, B. *Contribution pour une nouvelle Approche du dialogue Homme-Machine en CFAO*. PhD Université de Metz, 1995, 188 p.
11. Myers, B.A. User Interface Software Tools. *ACM Transactions on Computer Human Interaction*. 2, 1 (1995), pp. 64-103.
12. Paternò, F. A Theory of User-Interaction Objects. *Journal of Visual Languages and Computing*. 5, 3 (1994), pp. 227-249.
13. Patry, G. *Contribution à la conception du dialogue Homme Machine dans les applications graphiques interactives de conception technique : le système GIPSE*. PhD Université de Poitiers, 1999, 199 p.
14. Patry, G. and Girard, P. From Adaptable Interfaces to Model-Based Interface Development: The GIPSE Project, in *Proc. ERCIM Workshop on User Interfaces for All (UI4ALL'97)* (Obernai, France, 3-4 november, 1997), INRIA Lorraine, pp. 127-133.
15. Pierra, G. Towards a taxonomy for interactive graphics systems, in *Proc. Eurographics Workshop on Design, Specification, Verification of Interactive Systems* (Bonas, June 7-9, 1995), Springer-Verlag, pp. 362-370.
16. Puerta, A. The MECANO project : comprehensive and integrated support for Model-Based Interface development, in *Proc. Computer-Aided Design of User iterface (CADUI'96)* (Namur, Belgium, 5-7 June, 1996), Presse Universitaire de Namur, pp. 19-35.
17. Puerta, A.R. A Model-Based Interface Development Environment. *IEEE Software*. 14, 4 (1997), pp. 40-47.
18. Qiang, L., Wei, L., Ke, X. and Jiaguang, S. An Event-Driven and Object Oriented FrameWork for Human Computer Interface of CAD System, in *Proc. CAD & Graphics'97* (Shenzen, China, 2-5 Dec., 1997), International Academic Publishers, Volume 1, pp. 42-45.
19. Scapin, D.L. and Pierret-Golbreich, C. Towards a method for task description : MAD in *Working with display units.* Elsevier Science Publishers, North-Holland, 1990. pp. 371-380.
20. Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J. and E. Salcher. Declarative interface models for user interface construction tools : the MASTERMIND approach, in *Proc. IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction (EHCI'95)* (Grand Targhee Resort (Yellowstone Park), USA, 14-18 August, 1995), Chapman & Hall, pp. 120-150.
21. Texier, G. and Guittet, L. User defined objects are first class citizens, in *Proc. Computer-Aided Design of User iterface (CADUI'99)* (Louvain, Belgique, 1999), Kluwer Academic press, pp. in this book.