GDIT
Tool for the Design, Specification and Generation of Goals Driven User Interfaces

Antonio Carrillo, Juan Falgueras, Antonio Guevara
Dept. de Lenguajes y Ciencias de la Computación, Málaga University, España

Keywords: Human-Computer Interaction, Interaction styles, Design and Specification Tools, Automatic Prototyping.

Abstract: In previous works, we have made the proposal of, firstly , a new style of interaction called *Goals Driven Interaction* (*GDI*), specially appropriate for those software applications in which a zero learning time is necessary, and secondly , a methodology and a notation for the specification of these Goals Driven user Interfaces. Now, we introduce a software tool which brings support and facilitate the design and the specification of this type of user interfaces, using that specialized methodology and notation. In addition, the tool generates the corresponding prototype, from a model or specification made previously.

1 INTRODUCTION

Although *direct manipulation with WIMP* (Windows, Icons, Menus and Pointer) *elements* is currently the most extended user interface paradigm in use (Dix et al., 1998), there are still many users that need a training and learning period, manuals and/or expert support to become efficient users, and, in several cases, in order to be able to make the basic tasks. For this reason, in previous works (Carrillo et al, 2002; Carrillo, Falgueras, Guevara, 2005) we have proposed a new and alternative style of interaction, called *Goal Driven Interaction* (or *GDI*), specially appropriate for those software applications that are to be used seldom, and whose main priority is the ease of use and a minimal learning time.

GDI's main philosophy is to guide the user, in a hierarchical and progressive way through the whole interaction process, not only with regard to the tasks and goals that the user can have, but also about the steps sequence of the method to follow, or, if it is the case, the choices that can be made, to accomplish those goals at any given moment. To this end, the user interfaces based in this kind of interaction will be structured in three areas, as seen in Figure 1.

Goals Driver Window (GDW) will be the main mechanism for interaction in these interfaces, because it is the area where users will be guided gradually through the goals hierarchy (defined in the analysis and specification phase of the GDI interface), while allowing the user to access the different functionalities in the system, becoming a substitute (or alternative) to the typical menus, icons, toolbars and those elements in WIMP interfaces, that are not necessary in GDI (as seen in Figure 1).

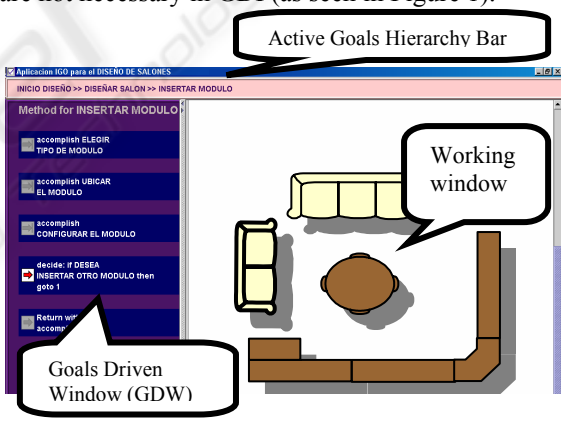

Figure 1: A "Goals Driven user Interface" (for the design of living rooms).

GDW will show the *Method* or the *Selection* that will allow the user to accomplish each concrete *Goal* at every moment. If that G*oal* requires the user to make a *Selection*, GDW will show the different excluding *Alternatives* that compose that *Selection*, so that the user could choose one of them. In other case, if the *Goal* must be accomplished following a specific *Method*, GDW will offer the sequence of *Steps* that compose this *Method* (as in Figure 1). The interface will always underline the current *Step*. And the system will only allow the user to carry out that *Step*. This could imply the initiation of a new sub-*Goal*, that will have associated another *Method* or *Selection* (whose description will be shown now). Or

Carrillo A., Falgueras J. and Guevara A. (2006).
GDIT - Tool for the Design, Specification and Generation of Goals Driven User Interfaces.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - HCI*, pages 135-138
DOI: 10.5220/0002496601350138

the *Step* could imply the performance of an elementary *Action* or activity. There is another special type of *Step*, in which the user *decides* if he wants to jump to a concrete *Step* of the actual *Method*, or to continue with the next *Step*. Finally, all *Methods* should end in a last *Step* that indicates that the *Goal* has already been reached or accomplished, and after which we will return to the 'father' *Goal* in the hierarchy.

On the other hand, two aspects are considered fundamental for the success of GDI and the interfaces that support them. The first of them is the fact that their specification and design process can be carried out based on main interface engineering techniques and methodologies, as those based on tasks hierarchical analysis (John and Kieras, 1996; Kieras, 1997b; Mori, Paterno and Santoro, 2002). Anyone of those techniques can be taken like reference. Nevertheless, even NGOMSL (Kieras, 1997a), the methodology (and the notation) more adequate for GDI, has needed to be adapted and extended, being converted in the *GDI methodology* already presented and described in (Carrillo, Falgueras, Guevara, 2005), whose basic elements we summarized now.

The generic format of a *Method* specification is:

```
Method for: <goal>     [ cancelable
  [disable_if <condition_for_system>]
  [effect <effect_on_the_system>] ]
   1) ...
   i) <step_i>
        [disable if <condition_for_system>]
        [effect <effect_on_the_system>]
   ...
   n) Return with goal accomplished
        [effect <effect_on_the_system>]
```

and each <step_i> (from 1 to n-1) could be one of:

```
accomplish <goal>
make <action>
decide: if <condition_for_the_user>
        then goto <step_#>
goto <step_#>
```

Selection has the following format:

```
[For the system>]
 Selection for: <goal>  [cancelable
  [disable if <condition_for_the_system>]
  [effect <effect_on_the_system>] ]
  a) <option_a>
    ...
  n) <option_n>
```

and any <option_i>:

```
 if <cond_for_user> [then accomplish <goal>]
  [disable if <condition_for_the_system>]
  [effect <effect_on_the_system>]
```

The *disable if* clause let to define *conditional Steps* or *Options* that will have to be disabled in case that the state of the system is the specified in the <condition_for_the_system>. Also, it will be possible to express *Steps* or *Options* that will have a concrete *effect* on the system, modifying their state.

A second fundamental aspect would be the possibility of using a software tool that would automate the process of generating a prototype. With this purpose we have developed the *GDIT tool*.

2 THE GDIT TOOL

The main aim of the tool that we present, is supporting and to facilitating the design and specification of new *Goals Driven user Interfaces*, using the specialized *GDI methodology*, and based on a model, generating the corresponding GDI prototype. Nevertheless, GDIT has a second aim. There are a few tools specifically designed to support building GOMS models, but they have not yet seen wide acceptance in any modeling or development community (Baumeister, John and Byrne, 2000). For this reason, and given the similarity between *GDI* and *NGOMSL* notations we have developed GDIT so that it is a good alternative tool for building GOMS models, using *NGOMS methodology*. Along next paragraphs we describe the structure (Figure 2) and the main elements of GDIT.

2.1 "Graph Window": The Visual Representation of a Specification

Since *GDI* and *NGOMSL* are essentially methodologies for hierarchical tasks analysis and specification, in our environment, models are structured in a hierarchical way.

Each project or model will be represented in the *Graph Window* by means of a (cyclic or acyclic) graph. Each node or component will map to, either a *Goal*, or to an elementary *Action*. Each *Action* is a leaf of the graph. There are another three kinds of nodes: the node that corresponds to a *Goal* that is accomplished following a *Method*, the node that corresponds to a *Goal* that implies a *Selection* and, finally, the node that doesn't follow any of the previous characteristics, the *indefinite* node. Indefinite node represents a *Goal* that, although it can be specified at an higher level of detail, the analyst considers that it is not necessary to be done in that moment. Each kind of node will be distinguished by a colour. In addition, for each node, further information can be given (clicking the node with the right button of the mouse), such as its type, the category, its specification, etc (see section 2.2).

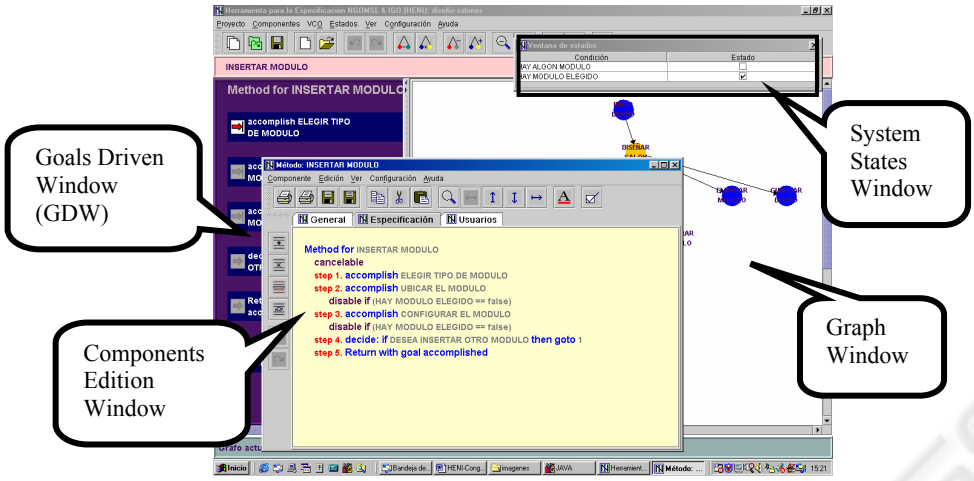

Figure 2: Main zones and windows of the GDIT tool.

The tool supports different views of the model: complete view, summary view, view by levels, view by types of nodes, enlarged or reduced view (zoom in and out)… GDIT supports the possibility of automatically expanding task patterns. This means that, if the designer defines the structure of a high-level goal in a point of the model, and if the same goal has occur somewhere else, then they do not have to again provide all the specification, but it is sufficient to indicate the name of the high-level goal.

The tool allows an easy copying and pasting of entire subtrees. It is posible to construct a library of patterns associated with subtrees for common tasks which the analyst can reuse in future models.

2.2 "Components Edition Window": Nodes in Detail

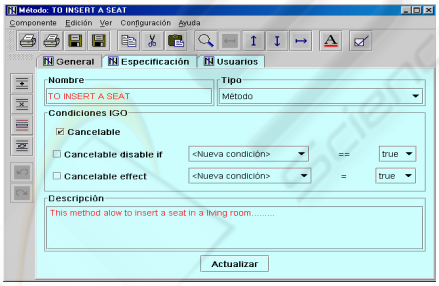

Figure 3: The "General data" panel of the "Components Edition Window".

1. Node details can be edited in a pop-up window (the *Components Edition Window*) that is divided in three panels corresponding to three cathegories: The first panel collects the *general data* of the node (Figure 3): name, type (*Action*, *Method*, *Selection* and *indefinite*, as we explained before), a detailed description, and the GDI clauses (*cancelable*, *cancelable disable if*, and/or *cancelable effect*).
2. The second one, called *Component Specification*, only will exist if the node represents a *Method* or a *Selection*. If the node is a *Method*, it gathers the list of *Steps* necessaries to accomplish that *Goal* (as seen in Figure 2). If the node represents a *Selection* it groups the several *Options* that compose it.
3. The third panel allows to view the list of 'fathers' of the current node. Nodes that use it or come exactly before it.

2.3 "Goals Driven Window": Edition and Simulation

There is another way for building the specification of a model, different from that of the Graph Window. Instead of that you can use the Goals Driven Window (GDW).

You can watch every *Step*/*Option* for each active *Method*/*Selection* in the GDW. Analyst can easily add, modify or delete *Steps*/*Options* in the active *Method*/*Selection* interacting with the mouse directly over those items. A contextual menu will help to do this edition.

On the other hand, GDW also can be use like a *interactive simulator,* to support the analysis, design and checking of the dynamic behavior of GDI model. This process is easy: you can click over the active *Step*/*Option* of the current *Method*/*Selection*, and then you will "navigate" over the hierarchy of *Goals* and *Actions*, as if a real process of interaction would be taken place. This will also allow you to check the process of interaction, and even to modify it, if necessary, in an interactive way. Designer can check whether the specified behavior is really what they intended to describe. This is important because,

especially in the case of large specifications, it is difficult to immediately understand the overall behavior deriving from the combination of the hierarchical structure and the temporal relationships.

2.4 The "System States Window"

GDI notation allows you to specify *conditional Steps*/*Options*, and also, *Steps*/*Options with effect* (by means of the `disable if` and `effect` clauses). These clauses need to examine or modify the current system state, which is represented by variables.

GDIT also includes a small status window in which, for the current project, those relevant states (variables and its values) are listed. We can see an example in the upper right corner of Figure 2.

2.5 Additional Features

At any time, it is possible to save partially or completely the specifications of the model, or to insert into the current model partial specifications previously saved. It is also possible to save or print all or parts of the specification as ASCII text and, also, all or parts of the Graph Windows as images. This is particularly useful when inserting them in documents, manuals, or reports related to the application that is being considered.

The tool also offers the possibility of changing the keywords of the GDI and NGOMSL notations. This makes it easy to localize the tool, making it more natural for people in any country.

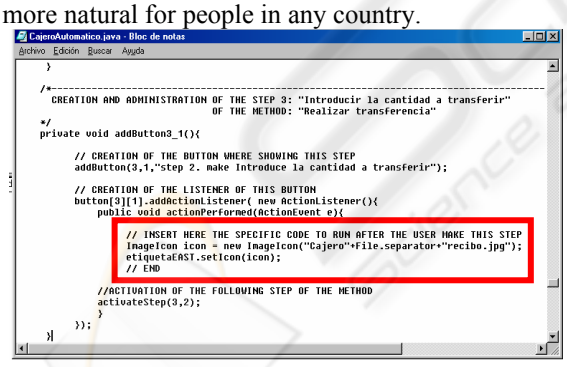

Figure 4: A prototype code piece generated by GDIT.

2.6 Automatic Generation of GDI Prototypes

Just after completing the specification of a GDI user interface, it is possible to ask for the automatic generation of Java code that corresponds to a real prototype of the application. GDIT generates the code of these prototypes in such a way that an external programmer could easily understand how the code is structured. For example, he could find quickly where to insert instructions to convert that prototype into a full-fledged application (Figure 4).

3 CONCLUSIONS AND FUTURE WORK

GDIT is a new platform that supports a new interaction style (GDI) and the corresponding methodology of development. It helps to carry out the specification and design of the user interface efficient and confortable. In addition, the development of both, a specifications language, and an automatic prototyping tool, has confirmed the viability of this GDI methodology.

One of the necessary further steps is the extension of our GDI specification language (and of our tool), building in it a higher semantic content. To this end a great deal of work is currently being carried out to incorporate the widgets characteristics as additional annotations in the specification of the models. The possibilities in this field are immense.

REFERENCES

Baumeister L., John B.E, Byrne M., 2000 : A comparison of Tools for Building GOMS models. *Proc. Computer Human Interaction Conf. (CHI'00),* pp 502-509.

Carrillo A., Guevara A., 2002: Goals Driven Interaction. *Proc. III congreso Internacional Interacción Persona Ordenador,* pp 68-75.

Carrillo A., Falgueras J., Guevara A., 2005: A notation for Goal Driven Interfaces specification. Raquel Navarro-Prieto & Jesús Lores, Interacción 2004, Springer, Dordrecht, The Netherlands.

Dix A., Finlay J., Abowd G., Beale R., 1998: *Human-Computer Interaction*. 2nd ed. Prentice-Hall.

John B.E., Kieras D.E., 1996: Using GOMS for User Interface Design and Analysis: Wich Technique?. *ACM Transactions on Computer-Human Interaction*, Vol3, No.4, December 1996, pp 287-319.

Kieras, D.E., 1997a. A guide to GOMS model usability evaluation using NGOMSL. In M. Helander, T. Landauer, & P. Prabhu (Eds.), *Handbook of human-computer interaction*. 2nd ed., pp. 733-766. Amsterdam: North-Holland.

Kieras, D.E., 1997b. Task analysis and the design of functionality. In A. Tucker (Ed.), *The computer science and engineering handbook*, pp. 1401-1423. Boca Raton, FL:CRC.

Mori G., Paternò F., Santoro C., 2002. CTTE: Suport for developing and analyzing task models for interactive system design. *IEEE Transactions on software engineering*,Vol28, No.8, August 2002, pp 797-813.