Computer-Aided Design of Menu Bar and Pull-Down Menus for Business Oriented Applications

Jean Vanderdonckt

Université catholique de Louvain
Place des Doyens, 1 - B-1348 LOUVAIN-LA-NEUVE (Belgium)
Phone: +32 (0)10 47.85.25 - Fax : +32 (0)10 47.83.24
E-mail: vanderdonckt@qant.ucl.ac.be
Web: http://www.qant.ucl.ac.be/membres/jv/jv.html

Abstract. Building a usable menu bar, related pull-down menus and submenus or cascaded menus remains an important design activity in the development of interactive applications, especially in the domain of business oriented ones. To provide some assistance to designers who are responsible for achieving this task, a two-phased design method for a menu bar and related pull-down menus is presented. Based on a entity-relationship model of the final application, a first phase automatically generates an initial menu tree; a second phase enables designers to interactively perform refinement operations on the initial tree to obtain a final menu tree. This tree can be finally exported to a graphical editor for free editing and adaptation. This method covers the selection and the positioning of menu items, a first proposal for mnemonics and accelerators that are intrinsically based of menu design guidelines.

Keywords. Automated Generation of User Interfaces, Computer-Aided Design of User Interfaces, Entity-Relationship Model, Guidelines, Menu bar, Model-based approach, Pull-down menu, Specification, Task model.

1 Introduction

Many aspects such as dialogue design, window or dialog boxes have been addressed in model-based approaches for user interface (UI), whereas the building of a menu bar, related pull-down menus and cascaded menus have received little attention. This situation is somewhat puzzling when the following are considered:

1. The usability of menu design is probably the most advanced in research and development, e.g. [5,6,9,12]. In particular, Norman has written a complete book about the trade-offs involved in menu design based on experimental research that have produced practical menu design guidelines [6].
2. Some techniques have been introduced to guide menu design in a structured way and various aspects such as the optimal number of items per menu [9], the traversal time of a menu, the depth and length of menu are well known [6]. Some techniques were developed to systematically produce usable menu structures manually.
3. Nearly all these contributions can be embodied in a software tool that automate parts of menu design. This could be very efficient since most relevant information to be exploited to significantly initiate a menu design can be found in traditional models (a task, a data or a domain model).

D. Duke et al. (eds.), *Design, Specification and Verification of Interactive Systems '99*

In a context where rapid application development is imposed by time and resource constraints, it could be appropriate to provide some assistance to designers who are responsible for designing a menu for an interactive application by developing a complete method for such a design.

The rest of this paper is structured as follows: section 2 will review the state of the art in the area of computer-aided design of menus by reporting on experiences on both methods and software tools; section 3 will present a method for producing a complete menu tree for a business oriented interactive application based on windows, icons, menus and popup (WIMP) technology; section 4 will highlight how this method can be applied on a small medical case study; section 5 will explain the complete design process supported by a software tool that works on a specification repository of models; section 6 will conclude by discussing some open issues resulting from the experience gained with this method and by summarising some future work.

2 Related work

MENUDA [10] was an experimental knowledge-based menu design assistant to aid the design of menu systems. Its knowledge base is structured as a semantic network of menu design guidelines collected from the literature in the domain of experimental research. Although this tool can identify relevant guidelines for each linguistic level (e.g., semantic, syntactic, lexical, physical), it does not produce any menu nor does it specify how these guidelines should be used.

MIKE [7] is probably the first research that followed a model-based approach. This software tool automatically generates a Unix graphical UI by exploiting the declarations of procedure signatures. These declarations, contained in the foreword of a Pascal unit, are exploited to automatically generate items on a menu bar and its resulting pull-down menus. The programmer writes the Pascal code of a unit containing the semantic functions of the application and adds comments for each signature from which MIKE will derive information to generate a menu. The signature

```
procedure Search_customer_by_id(customer,id,results);
          (* Menu='Customer' Item='Search customer by id.'*)
```

will generate a "Customer" item on the menu bar associated with a menu item labelled "Search customer by id."; all other procedures associated with the same menu bar item will appear in the same pull-down menu. In its Macintosh version, Mickey [8], the menus are generated in an equally straightforward manner.

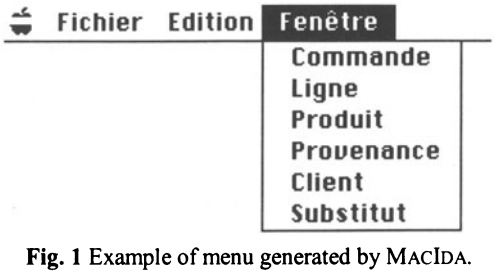

Fig. 1 Example of menu generated by MACIDA.

MacIDA [11] automatically generates MacApp code for a Macintosh graphical UI from functional specifications contained in an information structure model (graphically depicted as an entity-relationship model) and a dynamic model (graphically depicted as an activity chaining graph [2]). This tool extracts the names of entities and relationships found in the information structure model and generates a "Window" item on the menu bar containing these names associated with related windows or dialogue boxes (fig. 1).

JANUS [1] automatically generates a Windows'95 graphical UI from an object oriented model obtained from object-oriented analysis. The designer manually transforms this model into requirements stored in a JANUS Definition File (JDF) manipulated by a Computer-Aided Software Engineering (CASE) tool. This file is then exploited to map objects and their relationships (such as *is_a* or *is_part_of*) onto menu bar and pull-down menu items according to transformation rules (fig. 2).

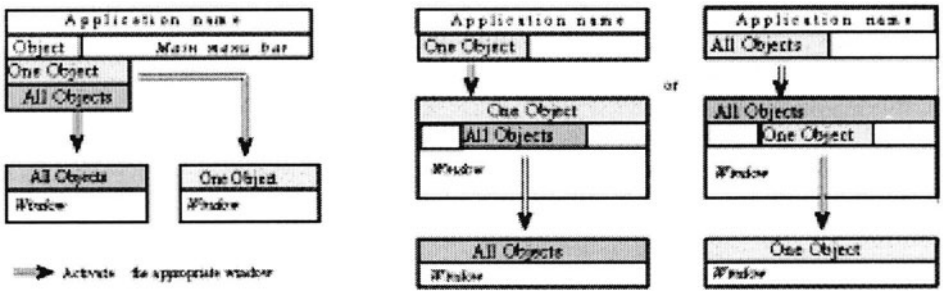

Fig. 2 Example of menu generated by JANUS.

These three last examples show that some significant parts of the menu tree can be automatically generated in a model-based approach where models result from software engineering. They are all based on the "object-action" paradigm frequently found in WIMP UIs: the task objects fill the menu bar as items and the actions which are performed on a particular object (or set of objects) fill the pull-down menus and sub-menus as items.

We can note that IBM *Common User Access* de facto standard is based on the "action-object" inverse paradigm, where the ordering of menu bar items is recommended as: `File, Edit, Actions, View, Options, Help`. Our proposed method will rely on the first paradigm, which is more commonly used.

3 A Method for Systematic Menu Tree Building

3.1 Hypothesis and definitions

As preliminary hypothesis, we assume that a designer has already performed some functional analysis resulting, for example, in an Activity Chaining Graph (ACG) as function dynamics model, which is itself deduced from a hierarchical task model [2], and an entity-relationship model as data model.

The set of items found on a menu bar, the related pull-down menus and sub-menus can form a menu tree structure whose leaf nodes are *terminal items* whereas all non-terminal

items are *selection items*. The user will be able to initiate an action as soon as he or she has selected a terminal item. Terminal items may appear at every level, although menu bar terminal items are not recommended [6], except if they are presented in a distinctive manner, such as `Item!`. Selection items are provided to guide the user in navigating through the menu tree and triggering semantic functions.

A *dialogue function* is a semantic function including input/output of information pertaining to the user's interactive task via interaction media such as screens, keyboards (for instance, "Search a customer by its identification"). A *service function* is an auxiliary function not directly appearing the ACG, but deduced from its configuration to improve the usability of the task (for instance, "List existing customers alphabetically" may not appear in an interactive task "Record an order for a customer", but can be deduced from its semantics). The *control* of a function can be *implicit* if it is triggered by the interactive application (for instance, when an order is placed in a pool, a summary of ongoing orders are periodically printed) or *explicit* if the user is responsible for its triggering (for instance, to record an address change in the customers' database).

A function with explicit control is said *global* when it can be triggered through any menu items wherever the use is in carrying the interactive task. It is said *local* if it is triggered in a local context which is only valid during a defined period of time in the interactive task (for instance, by a "Validate" push button in a dialog box).

3.2 Definition of the method

The method consists in traversing two phases: the first phase, producing an initial tree, can be completely automated by a software tool whereas the second should be computer-aided, thus providing designers more flexibility on the final tree.

3.2.1 Phase 1: Automated generation of a initial menu tree

1.1 Automated generation of a menu bar

Since the data model is considered to be a complete already existing entity-relationship model, we assume that all entities and relationships are known with their attributes. Although the rest of this paper will derive items based on entities and relationships properties, the strategy remains valid for objects and relationships properties from an object-oriented model serving as data model.

- Define E = set of all entities related to the application semantics.
- Define R = set of all relationships related to the application semantics.
- Sort entities in E by decreasing order of number of attributes, then alphabetically for entities having a same number of attributes.
- Sort relationships in R by decreasing order of main connectivity, then alphabetically for relationships sharing a same connectivity.
- Generate a first menu bar whose items are elements of E followed by elements of R. The items labels are the default names of entities and relationships found in the functional specification, with underscores removed and words uncapitalised.

The rationale behind these choices is that a menu bar should contain application objects which have associated semantic functions [13].

1.2 Automated generation of pull-down menus

- Define $P = P_E \cup P_R$ where P_E = set of interactive tasks (or phases in the application decomposition) sorted according the order of appearance of entities in E and P_R = set of interactive tasks (or phases in the application decomposition) sorted according the order of appearance of relationships in R.
- Sort P_E and P_R by order of function appearance as specified in the main ACG.
- For each entity or relationship in the menu bar, generate a pull-down menu whose items are the related interactive tasks. If there are no such tasks, no menu is generated. The labels of menu items are the default names of interactive tasks found in the functional specification with underscores removed and words uncapitalised.

Each interactive task consequently appears one time in P partitioned into P_E and P_R, while non-interactive tasks are not concerned. In these interactive tasks, functions with explicit control are particularly considered, whether they are global dialogue function or local service functions. The rationale behind these choices is that the menu tree should be organised according to the functional structure of an application and should reflect the user's task model [6]; all functions based on a user dialogue should appear in the menu tree. The ordering of menu bars items by order of importance is derived from the guideline stating that such an order is known to improve usability [5,6]. Basic actions should appear on pull-down menus [6]: therefore, interactive tasks that can be decomposed into dialogue functions will appear as such.

1.3 Automated generation of cascaded menus

	Level 1	Level 2	Level 3
Selection item	Attachment = pull-down menu Presentation = `Item`	Attachment = cascaded menu Presentation = `Item...`	Attachment = list of values Presentation = `Item∪`
Terminal item	Attachment = dialog box, secondary application window, function Presentation = `Item...`, `Item`, `Item!`	Attachment = dialog box, secondary application window, function, toggle Presentation = `Item...`, `Item`, `Item!`, `Item√`	Attachment = dialog box, secondary application window, function Presentation = `Item...`, `Item`, `Item!`
Abstract Interaction Object [3]	Menu bar	Pull-down menu	Cascaded menu
Labels	Names of entities and relationships	Names of interactive tasks	Names of dialogue and service functions
Structure	Guided by data model	Guided by Activity Chaining Graph	Guided by Activity Chaining Graph
Ordering	By importance	By sequence	By sequence

Table 1 Summary of design options for generating the initial menu.

- For each interactive task p appearing in P, define F_p in extension the set of dialogue functions and service functions which are not local.
- Sort each F_p first with dialogue functions sorted by order or appearance in the task ACG, then with service functions with no particular order.
- For each interactive task, generate a cascaded menu whose items are the F_p elements if $F_p \neq \varnothing$. The labels of cascaded menu items are the default names of functions found in the functional specification.

The rationale behind these choices is that actions and sub-actions of an application should appear in sub-menus [6,13] and preferably sorted by sequential order [6]. After performing these three steps, we conclude that each level 1 menu item is a selection item if there is a pull-down menu attached or a terminal item if there are not. Each level 2 menu item is a selection item if there is a cascaded menu attached or a terminal item if there are not. Each level 3 menu item is a selection item if a list of values is presented or a terminal item if there are not (table 1).

3.2.2 Phase 2: Computer-aided design of a final menu tree

2.1 Eliminating degenerated menus

At level 1 of the initial menu tree, we have *length*(level 1)= # items = # application objects (entities and relationships); we also have *length*(level 2)= # interactive tasks and *length*(level 3)= # non-local dialogue and service functions. Consequently, we can have at level 1 an item appearing in the menu which is attached with no pull-down menus if all tasks were attached to other items. Moreover, if a selection item at level 1 or 2 is attached with a single terminal item at the subsequent level, then the selection becomes no longer appropriate. The first item is then called *quasi-terminal item* and the menu is called a *degenerated menu*. The designer is then asked whether he or she would prefer to delete the terminal item to have a quasi-terminal item or to keep it as it is (fig. 3).

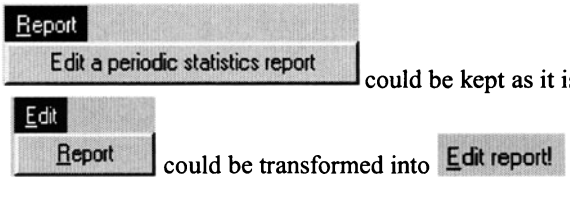

Fig. 3 Example of eliminating degenerated menus.

2.2 Unifying menu items

It is likely that two dialogue functions could lead to two different items in a menu. For instance, "Search a customer by its id." and "Search a customer by first name, last name" can generate multiple items in different menus. These two functions belonging to the same interactive task could be unified to have a unique item. On the other hand, two such items can be relevant from the user's task viewpoint. Thus, a confirmation is asked to the designer to see if two such functions should be grouped in the same task, thus appearing in a single item, or if a separation should be maintained.

2.3. Re-eliminating degenerated menus
This step results from the previous one: unifying two items into one can lead to a degenerated menu. In this case, it could be deleted with designer's confirmation.

2.4 Balancing the menu tree
Although the initial menu tree mirrors the structure and sequence of functions within interactive tasks, it is not necessarily balanced in terms of both depth and length. Kiger [5] recommends that the optimal tree depth is 2 whereas the optimal menu length is 8. A menu having few items, 2 or 3 for instance, could be embedded in a parent menu by replacing it with selection items assembled in a visual group delimited by two horizontal separators to denote the task visual disclosure. Balancing the tree consists in an iterative process from which a wider, yet less deep, tree is expected. Obviously, the resulting tree can become different from what has been derived from the ACG. It is thus required to ask designer's confirmation before applying any balancing operation. To guide the designer, the current depth and length could be provided.

2.5 Formatting the items
The labels of menu items are the defaults names of entities, relationships, and functions stored in the functional specification repository (for instance, "`Search_cust_id`" for "Search a customer by identification". Since each selection item should possess a meaning that mirrors the selection process and since each terminal item should mirror a task action, these default names are not necessarily appropriate or usable. Thus, the current label is reviewed for each item by the designer for editing and "..." is added for each terminal item attached to a dialogue box, "!", for each level 1 terminal item. Before editing, all underscores appearing in default names can be replaced by spaces under the designer's control, thus producing "`Search cust id.`".

2.6 Adding standard menus
According to the designer's need, standard menus can be selected and automatically added. Two alternatives are proposed: the IBM *Common User Access* norm and the Microsoft Windows standard. While the first promotes standard menus attached to a menu bar like `Workplace` or `Selected`, `Edit`, `Actions`, `View`, `Options`, `Help`, the second standard tends to prefer `File`, `Edit`, `Objects`, `View`, `Options`, `Windows`, `Help` or `?`. Theses standard menus are then added with all their regular and reserved mnemonics and accelerators.

2.7 Adding a Help menu
This feature is embodied to support the future implementation of a help system for the interactive application. If such a system will be implemented and if the Help has not been added during the previous step, a standard `Help` menu is then added. If no help system will be implemented, the Help menu is restricted to one item "`About...`".

2.8 Adding a `Quit` menu item
If a `File` standard menu has not been added previously, the item `Quit` responsible for quitting the interactive application is added as the last menu item of the first pull-down menu after a separator, as it is often the case [6].

2.9 Selecting the mnemonics
During this step, menu mnemonics are proposed to the approval of the designer: if an item is attached to a standard menu, then the standard mnemonic is selected (e.g., "Q" for Quit); if an item is not attached to a standard menu, the label initial is proposed. If this letter is already used as mnemonic in the same pull-down menu thus inducing a conflict, the next consonant of each word is proposed until the conflict vanishes. If such a conflict still exists after examining all consonants, the same process is reiterated on vowels and special characters, e.g., *,@,-.

2.10 Selecting the accelerators
During this step, menu accelerators are proposed to the approval of the designer: if an item is attached to a standard menu, then the standard accelerator is selected (e.g., Ctrl-P for Print). If an item is not attached to a standard menu, it would be appropriate for find relevant accelerators for the most frequently used functions. Since this information should come from other sources, only the functions located at the beginning of a task path on an ACG are subject to an accelerator proposal. It basically consists of a control key (Ctrl, Shift) along with the initial, a consonant or a vowel extracted from the label, preferably the mnemonic, if it has been determined before.

2.11 Including a menu tree map and help
When novice or intermittent users are interacting with the interactive application, a first guidance could be included as a map representing the menu tree associated with explanations for each item. This explanation is the default description of each task and function appearing in the ACGs. When the interactive application is used by experienced, frequent users, a list of keyboard or mouse accelerators could be automatically added as soon as the accelerators have been chosen.

3.2.3 Phase 3: Manual editing of a final menu tree

The final menu tree can finally be edited within the software tool itself: item editing, grouping, pull-down menus restructuring, item transferring from one pull-down menu to another. For this purpose, a three-view approach is followed (fig. 4):

1. An *individual view* displays all editable properties of each item, such as label, mnemonic, accelerator, activity,...
2. A *textual view* where the hierarchy of menu bar, pull-down menus and cascaded menus are displayed textually for fast item moving.
3. A *graphical view* where the menu tree can be directly manipulated as in a real UI.

In this manual editing, the designer is able to perform any traditional modification on the menu bar and the pull-down menus. This facility therefore provides room for defining customisable menus at design time. There is no support for modifying user-customisable menus at execution time.

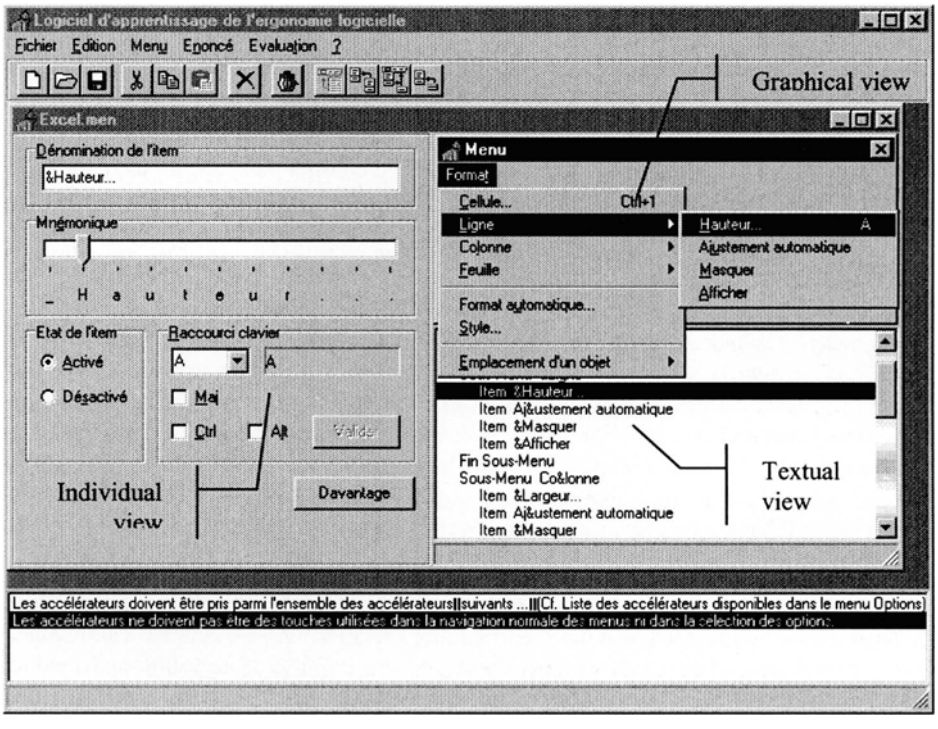

Fig. 4 Manual editing of the final menu tree.

4 An Example of Method Application

To exemplify the previous method, we apply it on a case study related to patient management in a hospital. From the entity-relationship model of this case study, we can extract E = {Patient, Care, Regimen, Room, Service, Hospital, Doctor, Prescription} and R = {Care_Prescription, Regimen_Prescription, Destination, Origin, Reimbursement, Room, Affiliation, Belonging, Movement, Perform, Is_part_of, Reimbursement_Prestation}. The first automated phase produces P = {Admission, Input_Transfer, Input_Exit, Prescription_Care, Prescription_Prestation, Cancel_Prescription, Recall_Prescription, Change_Regimen, Ask_Info_Patient_1, Ask_Info_Patient_2, Ask_ Info_Old_Patient, Select_Patient, Meals list}.

Interactive tasks are then distributed as follows:
$P_{E\_Patient}$ = {Admission, Input_Transfer, Input_Exit, Ask_Info_Patient_1, Ask_Info_ Patient_2, Ask_info_old_patient, Select_Patient}.
P_{E_Care} = {Prescription_Care, Prescription_Prestation, Cancel_Prescription, Recall_ Prescription}
$P_{E\_Regimen}$ = {Change_Regimen, Meals list}
$P_{E\_Room} = P_{E\_Service} = P_{E\_Hospital} = \ldots = P_{E\_Prescription}\} = \varnothing$

All other P_E are empty since their related interactive tasks have already been chosen. Similarly, all $P_R = \varnothing$. The decomposition of interactive tasks into functions gives:

$F_{Admission}$ = {Verify_patient_existence, Verify_patient_not_recorded, Verify_no_movement, Record_patient, Update_patient, Validate_doctor, Validate_service, Validate_ category, Validate_regimen}
$F_{Input\_Transfer}$ = {Validate_doctor, Validate_patient, Validate_source_bed, Validate_ target_bed, Verify_no_move, Verify_patient_in_bed, Check_free_beds, Record_ transfer}
F_{Input_Exit} = {Validate_doctor, Validate_patient, Validate_source_bed, Verify_no_ move, Verify_patient_in_bed}
$F_{Prescription\_Prestation}$ = {Validate_prescription, Cancel_prescription}
$F_{Cancel\_Prescription}$ = {Validate_prescription, Cancel_prescription}
$F_{Recall\_Prestation}$ = {Recell_prestation}
$F_{Change\_Regimen}$ = {Validate_doctor, Validate_patient, Validate_Regimen_ record_change_regimen}
F_{Meals_List} = {Meal_List}
$F_{Ask\_Info\_Patient\_1} = F_{Ask\_Info\_Patient\_2} = F_{Ask\_Info\_Old\_Patient}$ = {}

After accepting all default design options, the software tool produces the menu bar and pull-down menus depicted in fig. 5. The labels of menu items are the names of functions as stored in the functional specifications repository with underscores removed and names uncapitalised.

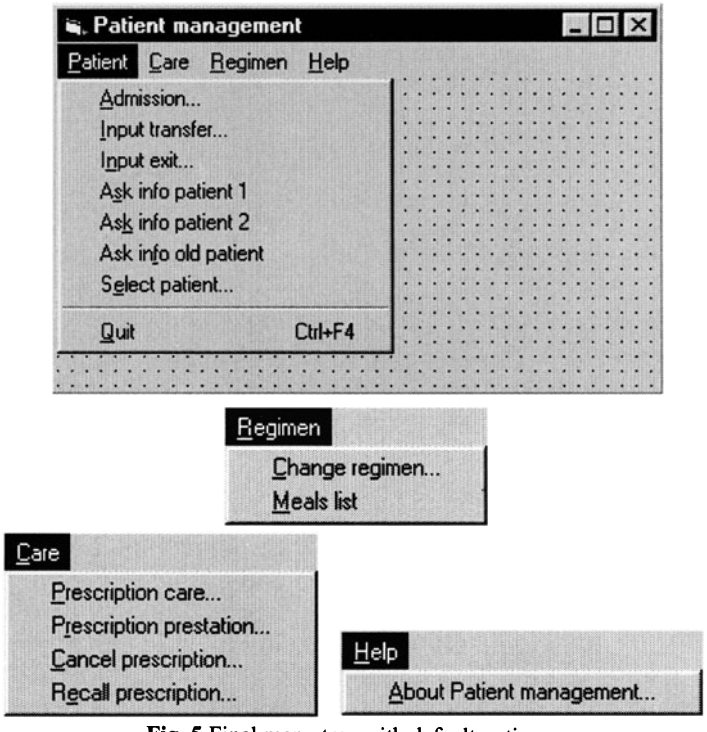

Fig. 5 Final menu tree with default options.

When all options are requested during generation, various facilities are added, such as the menu map and a list of generated accelerators (fig. 6). Of course, if these accelerators are edited manually outside the tool, the final menu tree is no longer consistent with the previous results. The designer is entirely responsible for checking that consistency is preserved and that no conflict is appearing. An inconsistency may appear between the initial model and the final menu as soon as a manual function introducing some inconsistency is operated on the menu tree.

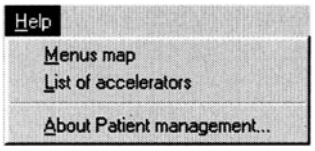

Fig. 6 Final help menu with complete options.

From these menus, it should be possible for the designer to easily change the labels of items, the mnemonics, the accelerators and the positioning of separators. Anyway, as much as possible is done to provide an initial menu tree that would be regarded as a good starting point for computer-aided completion. Designers are allowed to perform other manual operations such as (fig. 7): the changing of menu labels, choosing other mnemonics or accelerators, or introducing separators between groups of items.

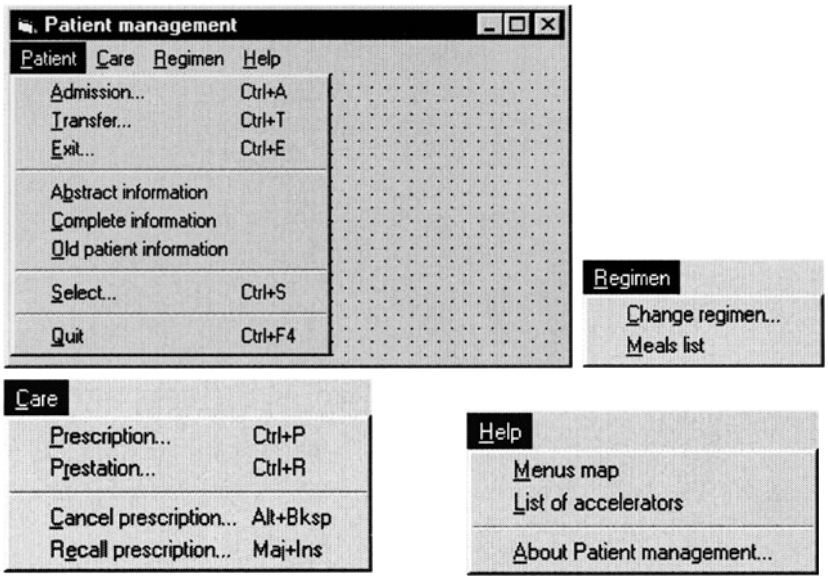

Fig. 7 Final menu tree after editing.

5 Tool Support for this Method

A software tool that scans a repository of specifications written in Dynamic Specification Language (DSL) [2] supports the above method. In this repository, various models are stored as textual declarative statements in different files: entity-relationship model, function structuring model, Activity Chaining Graph model... DSL is the related specification

language to declare these models. Models can consequently be input graphically with the model editor (fig. 8) which automatically produces the DSL specification from the graphical model or with the textual editor that directly produces the DSL specifications. The menu generator exploits these textual specifications to progressively generate a menu tree and to enable designers to interactively edit it within the tool. When manual editing operations are finished in the tool, it automatically (fig. 8):

1. Stores the resulting definition of menu as new DSL specifications for the presentation model [2]; it is expressed as instances of Abstract Interaction Objects [3].
2. Generates a .FRM file containing the definition of menus according to Visual Basic V5.0 format. This ASCII flat file can be easily manipulated since it contains concrete specifications like

```
VERSION 5.00
Begin VB.Form Essai
   Caption         =   "Patient management"
   ClientHeight    =   2550
   ClientLeft      =   165
   ClientTop       =   735
   ClientWidth     =   4980
   LinkTopic       =   "Form1"
   ScaleHeight     =   2550
   ScaleWidth      =   4980
   StartUpPosition =   3  'Windows Default
   Begin VB.Menu MenuPatient
      Caption         =   "&Patient"
      Index           =   1
      Begin VB.Menu ItemPatientAdmission
         Caption         =   "&Admission..."
         Index           =   11
         Shortcut        =   ^A
      End
...
```

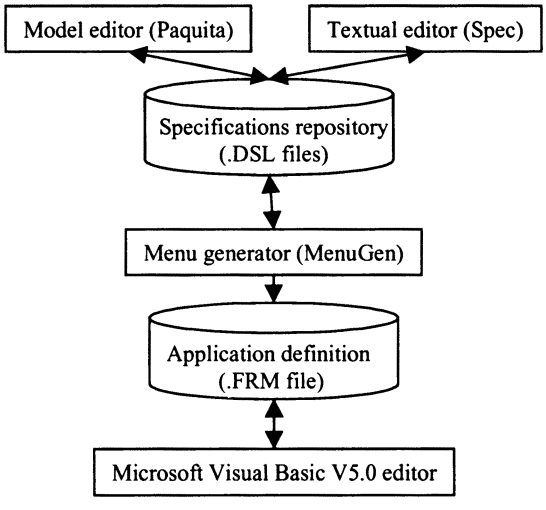

Fig. 8 The menu production process.

The designer is then able to perform any manual operations on the results within the Microsoft Visual Basic V5.0 editor (fig. 9) with the potential risk of loosing consistency with what has been previously generated in the menu generator. The menu resulting from this stage can be different with respect to the presentation model stored in the specifications repository, thus introducing some inconsistency. The designer is therefore responsible for updating the presentation model accordingly.

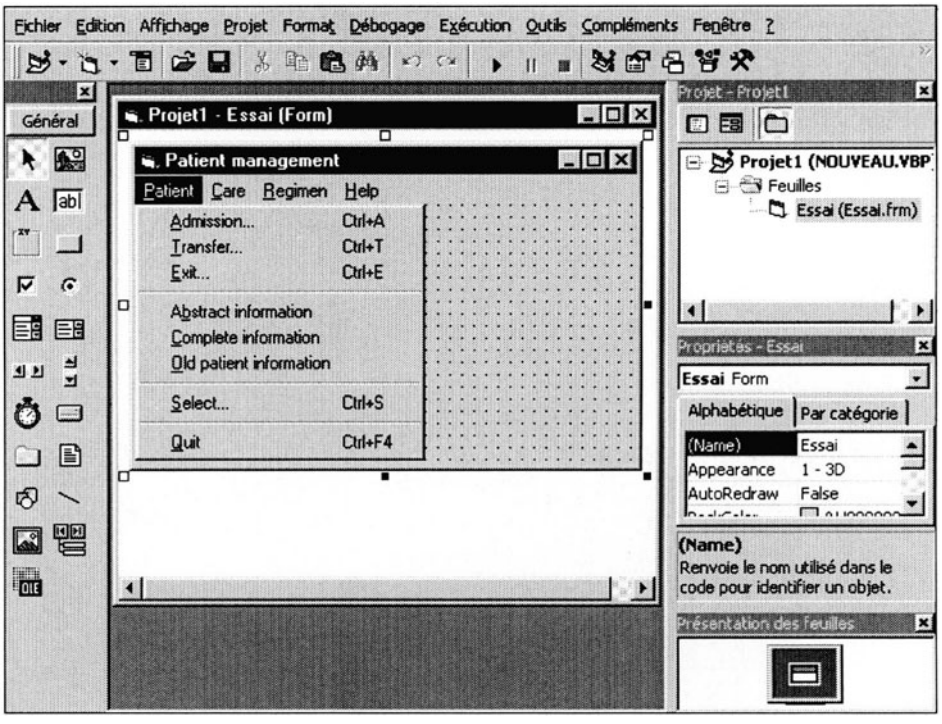

Fig. 9 Final menu tree generated for Visual Basic V5.0.

6 Conclusion

The proposed method for generating a menu bar and related pull-down menus is probably sub-optimal, but it seems impossible to propose such a method which uniformly and univocally leads to an optimal menu tree for all contexts of use. Moreover, determining the optimal menu for a particular interactive application must involve a range of activities, such as referring to guidelines, and user testing to guarantee that these guidelines is appropriate and that the results of the user testing are applied similarly.

What is interesting to note here is the underlying design options and guidelines that have been explicitly used for each step. Inevitably there is a selection process of these guidelines, and the problem of bias arises. Ultimately it has to be said that there are biases in this method, but that the guidelines upon which the method was created was described openly. Those biases are therefore discernible. The best protection against bias in this kind of method to have the process conducted independently by more than one team, and to be

subjected to expert review. Although we informally conducted such a process, complete method validity cannot be ensured. Many extensions are possible, depending on the adopted usability targets and the potential of translating existing menu design guidelines into operational rules and ergonomic algorithms. Seven possible extensions will be studied in the near future:

1. To look for new heuristics for positioning horizontal separators to visually identify blocks of related items into a group. For instance, such separators can surround menu items related to interactive tasks which can be grouped or which are semantically related to each other; moreover, similar heuristics might be created to group or ungroup sub-tasks according to the task model structure.

2. To design a command language that is syntactically equivalent to menus for frequent users (or to use some more sophisticated techniques to identify appropriate mnemonics and accelerators). Such research is already ongoing elsewhere [4,14]; fig. 10 shows how mathematical curve plotting commands in GNUPLOT can be typed after a command prompt or selected from the pull down menus to complete a command. In this case, the items selectable from the pull-down menus form a menu selection context which is completely equivalent to a command language. Each command can be initiated either by selecting it from the menus or by typing it as a command or both together.

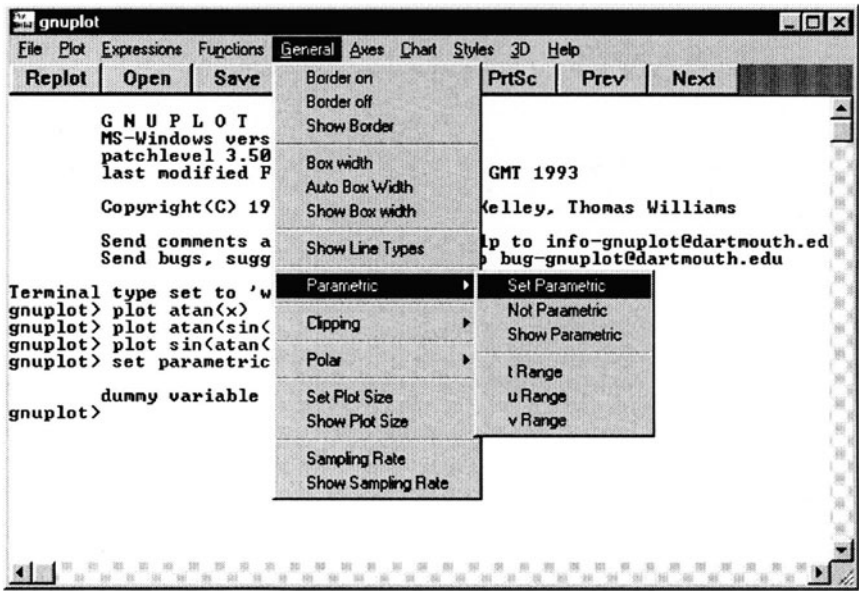

Fig. 10. Menu selection equivalent to a command language in GNUPLOT.

3. To generate a set of sub-trees when the number of items becomes critical: this situation particularly arises when the entity-relationship model contains many entities and relationships that are relevant to specific interactive tasks; the risk here is to generate a menu bar with too much items.

4. To define a variable UI by separating menu items into two categories: *fundamental items* which are frequently selected, processed or which are important for the application and *infrequent items* which are less frequently operated. This is aimed to introduce short and complete versions of a menu both novice and expert users, respectively.
5. To develop new heuristics for mnemonics selection: the mnemonics which are proposed in the current method are mainly based on a conflict resolution between letters without considering the letters in the menu items themselves. It is intended to develop heuristics which consider initial letters, phonetics aspects (especially for shortcuts) and significant letters (e.g., other letters than those which have poor underlining such as 'i', 'j', 'l').
6. To embody menu design guidelines which can be automatically tested on the current menu tree (e.g., guidelines for ensuring consistency, unique mnemonics and shortcuts, appropriate wording) to provide on-line critiques.
7. To explore how user-customizable menus can be supported: currently, all menu items are generated and determined at design time. To support user-customizable menus, a menu item should be generated and determined as the results of a run time procedure. A further analysis is required to determine how this kind of menu item can be generated in Visual Basic V5.0 and on what kind of model this generation can rely upon. For example, a simplified user model can be exploited to select a particular pull-down menu or any particular item depending on the user profile.

Acknowledgements

The author would like to thank Louis Simon, from Teleglobe Canada, for performing initial user testing of the method presented here, and Prof. Gilbert Cockton for his kind review of this manuscript. This work was partially supported by a belgian "FIRST-Université" research program, convention n°1407, supported by the "Région Wallonne".

References

1. H. Balzert, F. Hofmann, V. Kruschinski, C. Niemann. The Janus Application Development Environment-Generating More than the User Interface. In J. Vanderdonckt (ed.): Proc. of the 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96 (Namur, June 5-7, 1996). Namur: Presses Universitaires de Namur, 1996, pp. 183-206.
2. F. Bodart, A.-M. Hennebert, J.-M. Leheureux, I. Provot, J. Vanderdonckt: A Model-based Approach to Presentation: A Continuum from Task Analysis to Prototype. In F. Paternò (ed.): Proc. of 1st Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'94 (Carrara, June 8-10, 1994). Berlin: Springer-Verlag: Focus on Computer Graphics 1995, pp. 77-94. Available at http://www.info.fundp.ac.be/cgi-publi/pub-spec-paper?RP-94-023.
3. D.A. Duce, M.R. Gomes, F.R.A. Hopgood, J.R. Lee: User Interface Management and Design. In Proc. of the Workshop on User Interface Management Systems and Environments (Lisbon, June 4-6, 1990). Berlin: Springer-Verlag 1991.

4. W.K. Horton. Designing & Writing On line Documentation - Help files to Hypertext. Chichester: John Wiley & Sons, 1990.

5. J.I. Kiger. The depth/breadth tradeoff in the design of menu driven user interfaces. International Journal of Man-Machine Studies. Vol. 20. No. 2. February 1984, pp. 201-213.

6. K.L. Norman. The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface. Norwood: Ablex Publishing Corp. 1991.

7. D.R. Olsen. MIKE: The Menu Interaction Kontrol Environment. ACM Transactions on Graphics. Vol. 5. No. 4. Octobre 1986, pp. 318-344.

8. D.R. Olsen. A Programming Language Basis for User Interface Management. In K. Bice, C. Lewis (eds.): Proc. of the ACM Conf. on Human Factors in Computing Systems CHI'89 (Austin, 30 April-4 May 1989). New York: ACM Press 1989, pp. 171-176.

9. K.R. Paap, R.J. Roske-Hofstrand. The Optimal Number of Menu Options per Panel. Human Factors. Vol. 28. No. 4. August 1986, pp. 377-385.

10. K.A. Parng, V.S. Ellingstad. Menuda: A Knowledge-Based Menu Design Expert System. In M. Galer, S. Harker et J. Ziegler (eds.): Proc. of the 31st Annual Meeting of the Human Factors Society HFS'87. Santa Monica: Human Factors Society 1987, pp. 1315-1319.

11. I. Petoud, Y. Pigneur: An Automatic and Visual Approach for User Interface Design. In G. Cockton (ed.): Proceedings of the IFIP TC 2/WG 2.7 Working Conference on Engineering for Human-Computer Interaction EHCI'89 (Napa Valley, 21-25 August 1989). Amsterdam: Elsevier Science Publishers B.V., 1990, pp. 403-419.

12. P. Shoval. Functional Design of a Menu-Tree Interface within Structured System Development. International Journal of Man-Machine Studies. Vol. 33. No. 5. November 1990, pp. 537-556.

13. J.L. Sibert, J.D. Foley. User-Computer Interface Design. Tutorial #1, Conference on Human Factors in Computing Systems CHI'90. Seattle, April 1, 1990.

14. J. Whiteside, S. Jones, P.S. Levy, D. Wixon. User Performance with Command, Menu, and Iconic Interfaces. In L. Borman & B. Curtis (eds.): Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'85 (San Francisco, 14-18 April 1985). New York: ACM Press 1985, pp. 185-191.