FROM TASK ANALYSIS TO THE GUI APPLICATION DESIGN

I. Antcheva, B. Bellenot, R. Brun, F. Rademakers, CERN, Geneva, Switzerland

Abstract

One of the main application design challenges is the task of selecting appropriate Graphical User Interface (GUI) elements and organizing them to successfully meet the application requirements.

Questions to ask in answering this challenge, include: How to choose and assign the basic user interface elements, so-called widgets (from `window gadgets') into the single panels of interactions? How to organize these panels to appropriate levels of the application structure? How to map the task sequence to the different application levels? How to manage the hierarchy of the structure with respect to different user profiles? What information is absolutely necessary, what can be left out?

The answers to all of these questions are the subject of this paper.

USER GOALS, TASKS AND ACTIONS

In order to create an effective application it is absolutely necessary to gather user requirements; understand their goals and involve them as much as possible in all phases of the GUI design process. After collecting, organizing and evaluating user requests, the specific user tasks need to be clearly identified. This is achieved through task analysis. Before going into the task-analysis methods, four essential terms used in the following discussion are defined below:

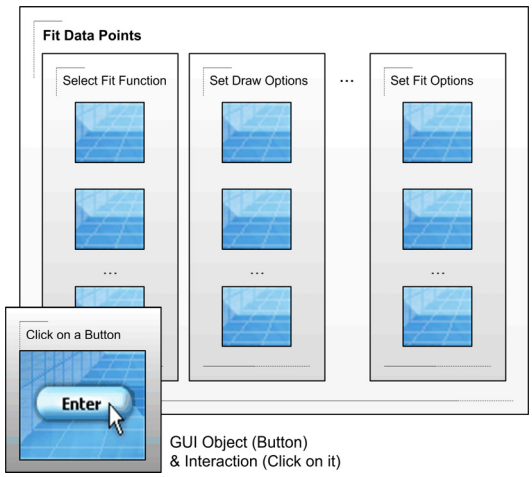

Figure 1: User actions, tasks and goals

GUI objects are the actual user interface elements (widgets and interaction devices) that pass information between users and the application. *User actions* are the operations initiated on GUI objects. *Tasks* are sequences of discrete actions to achieve a goal. *Goals are* the expected results when using an application.

The relationship between these items can be abstracted and presented graphically (see Fig. 1). The goal "to fit a set of data points" is divided into the following types of tasks: (i) select function for fitting, (ii) set fit options, (iii) set draw options, etc. Each task is further divided into user actions. An action is defined as a combination of a GUI object and the user interaction with it (for example, selecting specific options and parameters to perform a linear fit).

In general, users are prepared to divide a large-scale goal into several tasks and to accomplish these tasks one by one interacting with the application.

From the point of view of most users, an ideal automated application has a single-button operation and it would look something that like in Fig. 2.

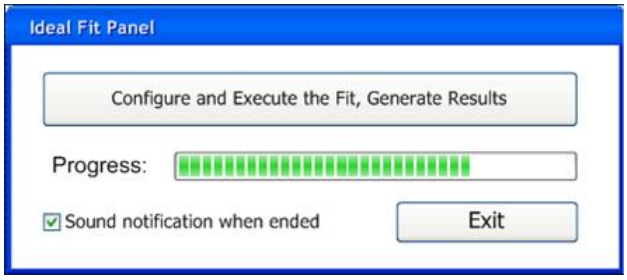

Figure 2: An ideal fitter application

However, in the real-world, users usually want to have a choice of several configurations, modify sets of valid parameters, monitor the process, specify different options based on the selected data before generating results, etc.

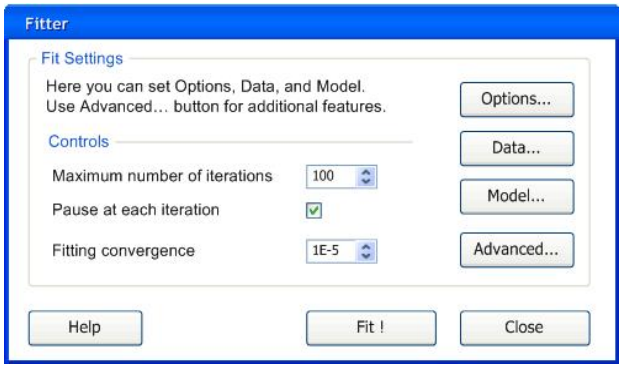

Figure 3: More flexible fitter application

The ideal automated application rarely provides an adequate level of flexibility and has to be transformed into a more flexible application (see Fig. 3). As a result, more GUI objects must be added to enable the desired wider range of flexibility and to allow a wider range of tasks.

TASK-ANALYSIS APPROACH

The purpose of task analysis is to map user requirements and to logically organize and distribute the GUI objects across the application. The challenge is to determine how far to go. On one hand, there is a potential to miss important objects and actions if there is a lack of detail in the analysis. On the other hand, the GUI will be unnecessarily complicated, if the analysis is taken too far.

The list of task analysis methods includes the following: hierarchical task analysis, use-case analysis, information-processing analysis, conceptual task analysis, graph task analysis, etc. Any of these may prove effective, although two methods appear to be most commonly used: hierarchical task analysis (HTA) and use-case analysis (also known as scenarios task analysis).

Hierarchical Task Analysis

The hierarchical task analysis uses a top-down approach to divide the application goals into constituent tasks, sub-tasks, and actions. Any action is a well defined combination of a GUI object together with user interactions.

Any application goal is presented as a list of tasks. Moving down one more level, different sub-tasks are specified with certain levels of precision. When looking at the breakdown level, it might not have sufficient details, and the analysis could break-down the sub-tasks further. Any additional level adds several more actions and a couple of GUI elements.

To keep all tasks logically organized, they have to be grouped and labeled by functionality:

- Each task must be included in at least one group.
- A task may also be common to several groups.
- A group composition may change.
- Tasks may be regrouped when it is appropriate.

Tasks within each group are organized in a hierarchical relationship [1]. Doing that, it is always useful to investigate alternative paths for the same user goal and to see if some steps can be eliminated, simplified or automated by implying so-called opening mechanisms. An Opening mechanism is a GUI object (a button) that opens a window to present a number of preset options or parameters. It gives all users the benefit of automation with the choice to override these values.

In fact, the hierarchical task analysis should continue to that level of decomposition, which defines all places of the opening mechanisms within the user interface hierarchy.

Use-Case Analysis

One popular alternative to the hierarchical task analysis is the use-case or scenario analysis. Using the film storyboard technique, it divides user goals by focusing on the limited number of tasks in a single use-case or scenario. Each scenario presents one complete operation where all steps are arranged as a sequence of actions, and it is clear where the operation starts and ends. A scenario can contain decision points, which promote and increase complexity. A decision point has two alternative sequences and can be presented as a question requiring a Yes or No answer

In a description of the scenario all nouns are potential GUI elements. For example, the scenario "*Logging into the system by providing a user name and password*" requires a panel with two text entry fields (GUI objects) for typing a user name and a password in addition to the usual set of buttons that confirm or cancel the user input.

If the user interface is designed by the programmers of a software system, there is a potential danger that scenarios can describe the internal functions of the software rather then users' task sequences [2]. Although it is important to know how the GUI ties into the underlying software system, the purpose of the use-case analysis is to define in human terms the sequences of task to be performed.

USER INTERFACE PRIORITIES

Next, after completing the task-analysis phase, GUI priorities must be assigned. Some GUI objects provide more important functionality than others and the application design should reflect this fact. In addition, these priorities depend on the different user classes that will share the same application. To simplify the explanation the GUI priorities for an application created for a single user class, are described below.

The safety of the application is of vital importance – if critical "stop" functions are required look at these first and provide "back-out" mechanisms and "error" indicators in order to make them the most accessible and responsive items in your GUI.

The next most important group includes items associated with the most frequently used tasks. They should be immediately available on the main panel, not more than one single click away.

In organizing the GUI hierarchy across the application sub-panels, GUI items have to be grouped for any given task and located on a single panel. Note, it is difficult for users to obtain information from another part of GUI when completing a task in progress. In addition, the logical task sequence needs to be taken into account in order to minimize the number of GUI management steps between tasks. Users can then foresee how one task naturally leads to the next.

The last thing to do is to find suitable metaphors. They can make the application easier to understand and use. Inappropriate metaphors limit its scope and confuse users.

If several user classes must share the same GUI, some of these priorities may conflict when the individual needs of novices, advanced beginners, competent performers, or expert users are considered and therefore need to be balanced [3].

An effective way to get out of this difficulty is to look first for information that is useful to most or all types of users. If the main application panel provides a common point for departure and prevents users from feeling that it

contains unnecessary or confusing GUI elements, the application will be accepted at first glance.

The remaining tasks for each type of user can be organized to class-specific sub-panels. At this point once again, the GUI elements for common usage should be presented consistently for different groups of users. Two or more types of users may need access to the same information to accomplish completely different tasks. With a little forethought, a single GUI panel can be designed for more than one type of user. Nevertheless, the specific needs of novices or advance beginners may put demands on the GUI that are inconsistent with the needs of competent performers or expert users. Rather then introducing additional panels, it is better to build separate applications to handle user class-specific tasks. This strategy offers two advantages: (i) removing class-specific GUI elements reduces application complexity; (ii) the GUI functionality can be easily designed for a customized main program according to specific user needs.

GUI PARADOX OF COMPLEXITY

Easy-to-learn and easy-to-use applications have a limited number of GUI elements distributed in a simple panel hierarchy. Users are satisfied immediately with a minimum effort. At the same time the application flexibility is sacrificed and users are not permitted to find new ways for applying the same application to new problems. The application serves the intended and limited functionality and it needs to be updated or re-written for any new user's strategy.

A data-analysis application or research tool needs to have flexibility and a less restrictive approach. For most applications, an appropriate design solution is somewhere between complete automation and the total user freedom.

An effective and helpful method serving a range of users is *progressive disclosure*. As its name implies, advanced features are exposed progressively in response to user interactions, e.g. by default it presumes that the most common user choices are presented first. Advancing deeper into the application interface, users discover more complex features at each step. This way the new users are not overwhelmed by a large number of sophisticated features; the experts have access to the functions and the application capabilities requested by their tasks.

The GUI panel hierarchy can be presented as narrow or wide (x-axis), shallow or deep (y-axis) 2D structures [4]. In easy-to-use applications the GUI hierarchy is both shallow and narrow – all GUI interactions take place in a small number of GUI panels.

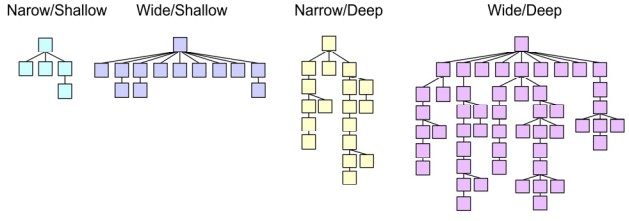

Figure 4: Different GUI hierarchies

The large applications have a wide and deep structure, which supports the complete range of its capabilities (see Fig. 4).

THREE-CLICK RULE

When designing the user interface of large applications, it is necessary to foresee the balance between GUI elements on each panel and the number of panel levels in the application depth. A useful rule to maintain focus on this point of balance is the *three-click rule* of design. This rule was developed by web site designers and reflects the primary goal, to get users to the information they try to find as quickly as possible.

The three-click rule simply means that every destination should be at most three levels deep, or three mouse clicks from the starting point.

The reason behind the three-click rule is related to the limitations of short-term memory (the memory of just presented). Information is retained in it automatically and can be retrieved without effort but the amount of this information is severely limited to 5-7 items. Without a conscious effort these items can be displaced by new sensory information (known as *Miller's law of 7* [5]).

Hence, if your application requires six mouse clicks, very likely the starting point will be forgotten before the user reaches the intended options or parameters. Therefore for such case it is better review the application design and try to reduce the depth of the hierarchy by reorganizing the GUI panel.

The three-click rule is a difficult target to reach for large applications, but it remains a lofty goal.

HOW TO MANAGE GUI COMPLEXITY

When developing large applications, it often seems there are a large number of parameters required for the user interface. Several ways help to reduce the quantity of GUI information.

In general, any application consists of a menu, working area and a status bar. *Menus* are the perfect progressive disclosure mechanism for the following reasons:

- They can provide accessibility to a large number of items from a single view.
- The menu items can be grouped by functionality with pre-assigned priorities allocated in so-called cascaded menus.
- They roll up when they are not in use.

If the number of items on your main user interface panel is large and starting to stress short-term memory, the simplest solution could be to reallocate a button function in a menu. This way the application offers access to the same number of functions and the depth of the GUI hierarchy is reduced.

Menus can be also be used to control user access to the application. In some cases user-level menus can let users personalize the application interface by giving them their own level of access to the application's functionality. For example, the application GUI can be programmatically designed to react to different user levels. If the user level

is set to Novice user, the complex GUI features can be hidden and the application interface is made appropriate for this type of users. By increasing the user level, users are permitted access to more advanced options of the application.

For applications providing system security the most practical solution is a password portal that assigns the user access level to the application functionality. That level controls which parts of GUI are displayed and accessible.

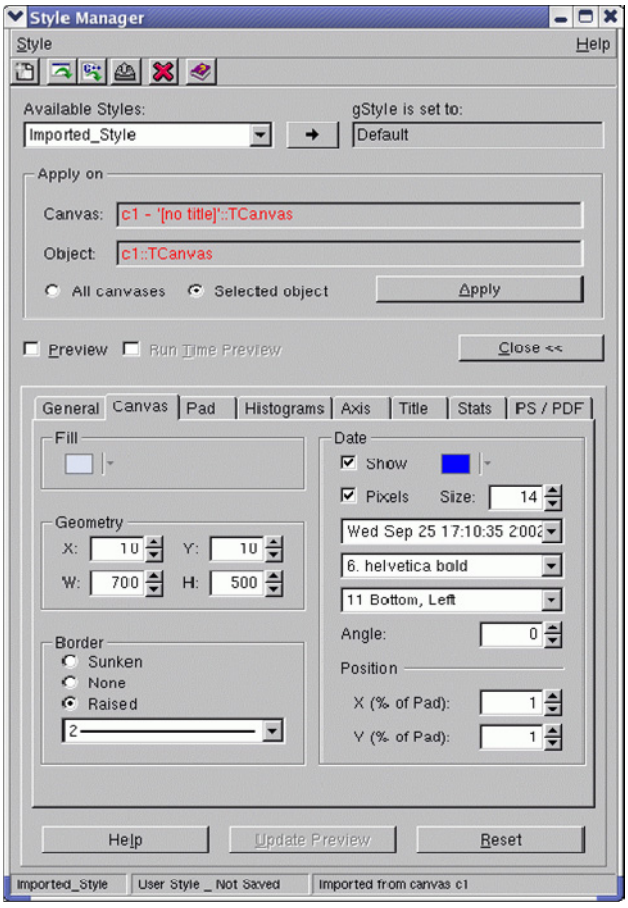

Figure 5: Style Manager in ROOT uses Tab controls for grouping options and parameters in several panels

Tool bars have become a standard feature in many modern user interfaces providing immediate, top-level access to important functions. They contain buttons for the most frequently used functions. In comparison with menus, they avoid the extra click needed for opening a menu. Menus require visual scanning and reading to locate the item of interest, while in contrast, tool bar buttons use pictures and icons and do not oblige reading. In any application tool bar buttons can be used as short cuts for the menu items.

Another useful progressive disclosure mechanism is the *Tab control*. It allows several panels to co-exist in a single frame (see Fig. 5). Users have one-click access to a large number of parameters and options in a single panel and do not need to open and close different windows. The metaphor of Tab controls is based on a notebook with tabs used to separate sections.

According to the *Fitts' law:* the time to get a target is a function of a distance to and the size of the target [6]. Put simply this means smaller GUI targets as tiny buttons will slow down users and reduce their productivity. Also, the more information presented on a single panel minimizes the relative importance of each GUI element.

There are three ways for reducing the information overload on a GUI panel.

1) Removing unnecessary and redundant elements by keeping only those, which are tied directly to user goals determined by the task analysis.

2) Hiding some of the GUI elements and showing them when necessary.

3) Presenting large amount of information in more efficient ways by:

- Grouping elements and removing unnecessary repeated labels.
- Having tool tips where possible – note when used in combination with picture buttons they conserve a lot of space.
- Using pictures and icons to eliminate the need of labels as visual metaphors.

CONCLUSION

This paper is focused on collected knowledge and specific experience that have been used in the GUI design process in the ROOT project [7]. Feel free to adapt and expand any of the shared ideas for your own projects as we believe that designing effective user interfaces is the key for the success of any project.

REFERENCES

[1] Roberts et al., Bridging User Interface Design and Software Engineering, McMillian Tech. Publishing, 1998

[2] Weinschenk et al., GUI Design Essentials, John Wiley & Sons, 1997

[3] Antcheva et al., Guidelines for Developing a Good GUI, Proc. CHEP04, 2004

[4] Norman D.A., The Psychology of Every Day Things, Basic Books, 1988

[5] Miller G., The Magical Number 7, Plus or Minus Two: Some Limits of Our Capability for Processing Information, Psychological Review 63, 1956

[6] http://www.asktog.com/

[7] http://root.cern.ch/