CONTROL SYSTEM APPLICATION LOOK AND FEEL GUIDELINES

M. Kadunc[*], J. Kamenik, I. Križnar, G. Pajor, M. Pleško, A. Pucelj, I. Verstovšek,
J. Stefan Institute and Cosylab Ltd.

Abstract

Control system applications, usually developed using modern RAD tools, are often inconsistent and do not meet the visual or functional requirements of the users. To avoid such problems, Cosylab[1] adopted a set of guidelines for writing good user interfaces. Most of these guidelines were gathered by studying various user interface approaches in control system and common PC applications[2,3], identifying specific problems and finding the most acceptable solutions. This paper briefly discusses the choice of a color scheme for control widgets, labels and other components often used in control panels, as well as their layout on screen or in a window. Font, case and language of different texts used in applications are also suggested. Furthermore, it focuses on the interaction between the user and the computer, first by enumerating and evaluating ways of displaying messages to the user, and then by addressing accessibility issues, such as keyboard shortcuts, contextual (pop-up) and window menus, toolbars etc. Finally, it addresses the customization of application workspace and the problem of finding a balance between user-specified and default look and feel of the applications.

1 INTRODUCTION

When a new control system is being developed, most of the time is spent programming the server architecture, integrating the hardware equipment, writing drivers, network protocol, client access libraries and application programming APIs. While all of these are very complex pieces of software, writing applications and especially designing graphical user interfaces is usually considered to be a simple task, done either by physicists who are not professional software developers, but rather the intended users of the applications, or by programmers who otherwise have more "important" things to do, but have to write the applications because the operators really need them. Many people end up writing applications, each with a different sense of design and a different definition of user-friendliness. On the other side, there are the control system operators, who will use the applications daily and thus ask for consistent, intuitive, easy-to-use, appealing and, if possible, customizable look and feel of the applications.

At Cosylab, we defined a set of guidelines for writing user interfaces and encouraged our developers to use these guidelines. We appointed a single person to be responsible for the GUI look and feel, inspecting the work done by other programmers, suggesting improvements and helping them write good user interfaces. We also decided to design a set of useful GUI widgets, stored in a common repository and accessible by anyone writing applications. The widgets are designed to be consistent with the Java™ metal look and feel, using standard colors, fonts, borders, keyboard shortcuts and mouse actions. They are intended to minimize the effort needed to design and implement an application user interface.

2 VISUAL DESIGN

2.1 Colors

Use a simple color model, with no more than two different basic colors for default functionality and a standard set of colors to communicate additional information to the user. Background and borders of components should be rendered using light, non-saturated colors, which do not distract the user and are natural to the eye. These are typically shades of gray. Selected or focused items should use a more saturated, neutral color such as blue. Text should be rendered using black and should have a white background when editable.

Other colors should have a special meaning and they should be used consistently across applications. Most common colors are: strong red for errors, green for start, yellow for warning, etc.

An application may define its own colors with an associated meaning to clarify the user interface and identify certain items, but no more that seven different colors should be used, otherwise the users will have difficulty remembering their meaning. Use darker, saturated colors, but make sure that they are not too similar to the standard colors mentioned above.

2.2 Application Graphics and Animation

Icons, button graphics and symbols are very useful to visually represent an action or an item without using a large area of the application. Graphics in applications should be very simple, easy to remember by the users and should identify clearly the objects or concepts they represent. A "family" of icons that identify similar actions or concepts should utilize common visual elements to separate them from other groups of icons. Color rules for images are not as strict as for the rest of the application controls, but you should nevertheless use a limited set of colors and avoid large areas of saturated colors, such as pure red or pure blue.

Animation can provide effective emphasis if used correctly, but a GUI designer should give careful thought whether an animation is appropriate. Limit animations only to situations where they provide meaningful feedback to the user, do not interfere with the user's work and do not detract the user from more important screen elements.

*miha.kadunc@cosylab.com

2.3 Fonts and Text

Text is an important design element and appears throughout the application in labels, buttons and other components. To ensure consistency, ease of use, and visual appeal, only one simple font style in no more than three different sizes should be used for the whole application. For most applications, only one font color (preferably black) should be sufficient. Other colors should be used to help the user identify certain values or to notify the user that an unexpected situation appeared.

Language used in applications should be clear, concise and consistent, with wording that is readable and grammatically correct. Use headline capitalization for most of the names, titles, labels and short text, sentence capitalization for lengthy messages. Only one language should be used otherwise users might get confused.

2.4 Borders and Layout

Borders are commonly used to visually separate logically different parts of an application. Similar borders are also parts of components used in applications. Application designer should use a single style of borders throughout the application. The borders' color should be similar to the component background color, different enough for the user to notice the border, but moderate enough not to distract him from important data. Borders should be used as an orientation only, not to impress the user with colors and shapes. One should use borders only to make an application clearer to the user.

GUI designers should give careful consideration to the layout of components in windows and dialog boxes. The best designs are aesthetically pleasing and easy to understand, enabling the user to move through an application and utilize its features effectively. Components should be laid out in a logical order, with more important components in the center, on top and to the left of the application, less important components at the bottom and to the right. The main application window should only contain components that will most probably be used by the operators during normal operation. Additional functionality and optional application features should be available via application menus and displayed on demand.

The layout should maintain its visual appeal and functionality if the application window is resized. In most of the graphical environments, there are many components that facilitate creation of resizable layouts, such as scroll panes, split panes, tabbed panes, layout managers, utility windows, internal windows and other. With the use of such components, users have more control over the appearance of the application and can enlarge or expose individual areas that they are interested in.

2.5 Menus, Toolbars and Tooltips

Menus have to be well organized so that the operators can navigate them intuitively. Thus it is best to implement standard menus like File, Edit, View, Help, etc, which most users are already familiar with. These primary menus positioned in the menu bar should be equipped with keyboard mnemonics (ALT + letter) for faster access. Item names should be capitalized and one word in length to be easily recognized. Secondary menus' names can be longer. Menu options containing submenus should indicate this with an arrow displayed next to the name and menu actions invoking a dialog should indicate this by displaying three dots ("…") next to their name. If a toolbar button exists invoking the same action as a menu option, the icon of the toolbar button should be displayed on the left of the menu option to indicate this. Finally, a separator line should separate groups of similar actions in a menu.

Popup menus can be very useful for configuring and performing actions on individual components in an application In principle all actions from popup menus should also be accessible through the main application menu bar.

Toolbars should be placed at the edges of the application and should take up 20 to 30 pixels of screen height. All toolbar actions should also be accessible through application menus while all buttons on a toolbar must be equipped with a descriptive tooltip text.

Tooltips are very useful since they educate and inform users about the functionality of the application. When using tooltips, buttons can be stripped of all text and be distinguished by icons only. New users read the tooltips to learn the functionality of the corresponding buttons while experienced users recognize buttons by their icons and can ignore the text.

3 BEHAVIOR

3.1 Messages

The user has to be informed about any event that occurred inside an application without his direct initiation. Actions taking considerable time to complete should indicate their completion status via an hourglass cursor and a progress bar when appropriate and meaningful. The user also has to be notified upon their completion. Expected and frequently occurring messages can be displayed in a status bar or preferably in a special textual report area, while unexpected and important messages should be displayed in a special message box. If the operator attempts to perform an action that would seriously affect the application or the control system, a warning window should appear allowing confirmation or rejection of the attempted action as to prevent accidental misacting. Finally if the startup of an application takes more than a couple of seconds, a splash screen should appear possibly displaying the current status of application initialization. It is advisable that all messages be saved in some log that the user can later inspect.

3.2 Dialogs and Wizards

Dialogs and wizards should be accessed through menu options marked with three dots (e.g. "Options…"). The first are to be used for configurations and more important, less frequently used actions, while the second are used for complex tasks that can be split into smaller steps and presented to the user one step at a time.

A typical dialog is of fixed size and contains a few text fields, check boxes and radio buttons together with a pair of control "OK" and "Cancel" buttons ("Apply" may be included if appropriate) in that order. The control buttons should be placed in the lower right corner of the dialog, while the design of the rest of the components above them should follow the same guidelines as the rest of the application. The "OK" button should be the default command that can also be accessed through the "Enter" key on the keyboard while pressing the "Esc" key should be equivalent to the "Cancel" command.

The layout of a wizard should be similar to that of a dialog: control buttons "Back", "Next", "Finish" and "Cancel" should be placed in this order in the lower right corner. The "Next" button should only become enabled after all critical parameters have been validly set in a panel, while the "Back" button should always be enabled to allow user to correct parameters set in previous panels. Pressing the "Finish" button should set all the parameters on remaining panels to their default values and complete the wizard.

Dialogs and Wizards should be modal, i.e. they should block access to other parts of the application. If there is a good reason for not doing so (when application should be responsive while the dialog is displayed) they should be displayed on top of the application.

3.3 Mouse and Keyboard Actions

Mouse actions should be used for switching focus between parts of application, selecting text or other elements, operating graphical controls like buttons, check boxes, etc… and drag and drop (d'n'd) operations. In general, keyboard equivalents should be implemented for all mouse operations inside an application.

The following keyboard shortcuts should be implemented in all applications: The TAB key should be used for switching between components (Shift + TAB in textual components where TAB has a different meaning) and ALT + letter for activating individual components with mnemonics. Inside a component or menu the arrowed keys should be used for cursor, focus or selection movement. Keyboard shortcuts for copy (Ctrl + C), cut (Ctrl + X) and paste (Ctrl + V) operations should be implemented where appropriate. The Space key should be used for activating buttons, check boxes and radio buttons in focus and for selections while Ctrl + Space should be used for adding selection and Shift + Space for range selection. The Esc key should be used to dismiss a menu or a dialog box without changes or to cancel a d'n'd operation in process. The Designer of the application should make sure that no implemented shortcuts in any context inside the application interfere with any existing shortcuts in any operating system the application is designed to run in.

4 DESIGNING WIDGETS

Individual graphical widgets are usually not stand-alone end-user products but are designed to be used and reused in different contexts inside an application and even in different applications, thus great care has to be taken in their visual design to reflect their reusability and adjustability. Standard colors should be used in all widgets so that they cohere well with the rest of the application. The designer of a widget should focus on details, especially insets and borders, as they dictate how the widget will connect visually to other components in its context.

Another important aspect of its visual design is the widget's adjustability to different sizes and different user (programmer) needs. If possible, different visual and functional modes should be implemented to be switched between in different contexts. A component used for displaying numbers could have a simple mode showing only the number, or an extended mode which would also display the maximum an minimum values, units, average etc. Still it is important for the widget to retain some of its common characteristics and that it is easily recognizable in all of its modes

The designer should implement support for as many ways of controlling the widget as possible. Keyboard shortcuts for switching focus inside the widget, standard action keys, arrow keys for moving etc. should all be mapped. Mouse buttons and mouse wheel should be put to some use if possible. Furthermore d 'n' d and copy-paste support for values and texts should be implemented.

Tool tip text should be applied to all buttons and other active components of the widget.

5 CUSTOMIZATION

Use customization when you consider that the application's functionality might be increased if the users specify their own appearance and behavior. Users should not be bothered with every detail of the application's look and feel, such as picking appropriate colors, moving the components, specifying shortcuts etc. - this is the work of the application designer. They should only choose between a small number of relevant options, all of which should produce a consistent, appealing and functional look and feel. The configuration should be persistent, allowing the users to save their settings and load them the next time they use the application.

6 REFERENCES

[1] http://www.cosylab.com/

[2] Sun Microsystems, Inc., "Java™ Look and Feel Design Guidelines, Second Edition", Addison-Wesley, February 2001.

[3] D. Springgay et al., "Eclipse User Interface Guidelines", Object Technology International and others, February 2002