Host Frame User Interface and Its Architecture

MEHMET S. UNLUTURK KAAN KURTEL COSKUN ATAY
Department of Software Engineering
Izmir University of Economics
Sakarya Cad. No.156 Balcova 35330 Izmir
TURKEY
suleyman.unluturk@ieu.edu.tr http://www.ieu.edu.tr

Abstract: - User interface research has shown that the user's primary focus is the center of the screen. The periphery is available and often used to contain elements that are helpful but non-essential to the specific existing goals. Strong present examples of this can be found in Microsoft's Multiple Document Interface Office products (such as MS Word and MS Excel). Microsoft Outlook and Lotus Notes are good examples of products that are more in tune with multiple application interfaces rather than just multiple instances of similar documents. This paper is following many of the elements set forth in those examples, as significant marketing research has been incorporated into those products. User personas of window's applications range from sales demo's to trade show viewers to beginning users to expert users. The PC host of these can be touch screen or mouse driven, and the display itself, while typically standard VGA variation, may also go to a wide screen format for some applications. The frame / application combination allows variations to create an optimum experience for each of these scenarios. In addition, this paper introduces the enhanced status line by applying a basic user interface design principle of "combine display and update capabilities" to the status line of a host frame.

Key-Words: - Interface, Multiple Document Interface, Polymorphism, Status Line, User Interface, Window DLL.

1 Introduction
The following common elements (see Table 1) should appear in every application. If they can be part of the frame, they do not have to be specified and designed multiple times (for each application) and the spatial relationship remains the same for each application, providing a familiarity as the user moves from application to application.

Table 1: Common Elements

Company Logo	Must appear somewhere
Facility Logo	Optional for user
Help	Optionally always visible is significantly greater benefit than a user selected dialog
Application Name / Type:	Critical as we allow application switching
Application Selection	Menu, dropdown list, toolbar, and/or button selection of foreground application
Operation Selection	Menu, dropdown list, toolbar, and/or button selection of these applications major operations
Wizard Buttons	Cancel, Back, Help, Next, Finish generic buttons for any wizard-style operation sequence
Application Status Line	Any problems with the current state of this application
Login Name	Who am I logged in as?
System Status Icon	Is the system feeding this application currently fully functional?

In this paper, common elements that are given above are defined as interfaces. These interfaces describe only the signature of properties and methods, while applications that are hosted by the host frame provide actual code for those properties and methods as necessary. The code in each property and method can differ from application to application, given the semantics of each method is protected. Each application can implement the same method in a different way is the basis for polymorphic behavior [1, 2].

Host frame places these common elements around its client area that is the middle of the screen (see Figure 1). Host frame loads each selected application inside this client area as long as the selected application implements the interfaces for the common elements. During the loading process, host frame changes its caption, help content (visible to the user), menu items with respect to each foreground application. These requests are received through software interfaces. Next section details how the frame does customization base on user's computer skills.

2 Frame Customization
Frame customization (a set of view properties) is set up from the menu under a VIEW option. Each login can create its own view property preferences, and has exclusive access to change them. A password (the same as the login password) is required so that users of 24/7 applications (which auto-login) will not be able to

change the view properties, unless specifically authorized through access to the password.

Customization is encouraged by making the process as simple as possible, and by letting the user work with the changes before deciding whether or not to save them. The application status bar warns the user of unsaved changes to the view properties, and it is easy to either save them if the user likes them, or to restore default settings or user's previously saved settings. In addition, it is simple to import settings saved from another login. This lets the administrator create a unique look, and let each login use that as its baseline, and customize from there.

All the view properties are stored in the database for each login. Regardless of what machine user logs into, user's application frame will look the same.

The suite of applications hosted by the frame is dependent on the login privileges granted by the administrator. Each can be accessed via application lists (menu, dropdown list, and toolbar) or via buttons. Default lists include all the privileged applications, plus some formatting: the applications are grouped into categories, a category title line precedes each category, and a blank spacer line precedes each category title. The application list itself begins always begins with the first line "Choose another application" and the second line a blank spacer. Any item to appear in the application list can be disabled if a particular login will not be using it (except the first two lines). This includes disabling category names and spaces if the user believes the application list is clearer without them. Disabling items from the application suite list affects the applications menu, the dropdown list, and the toolbar.

Application buttons are also available (up to 7) for touch screen switching between applications and to show off the applications available. Any application (but not categories and spacers) is available to appear as a button independent of its enable/disable state in the list. Button disables work similar to list disables it. Application buttons should center in the application button bar if less than the full 7 is enabled.

Text can be changed on the buttons and lists to match a facility's specific terminology. Text changes appear on the buttons, dropdown list, and application menu, plus wherever the application name appears (header, caption, view menu). The 1 to 2-character toolbar abbreviation text should be changed as well to match the full text. Spacing between words in the custom text can force the words to 2 lines on the buttons, but should be condensed in all other displays of the text.

Elements of the screen can be turned on and off to customize the screen for different user personas, depending on the goals and experience levels of that particular user, as well as the hardware (touch screen) available. Options include

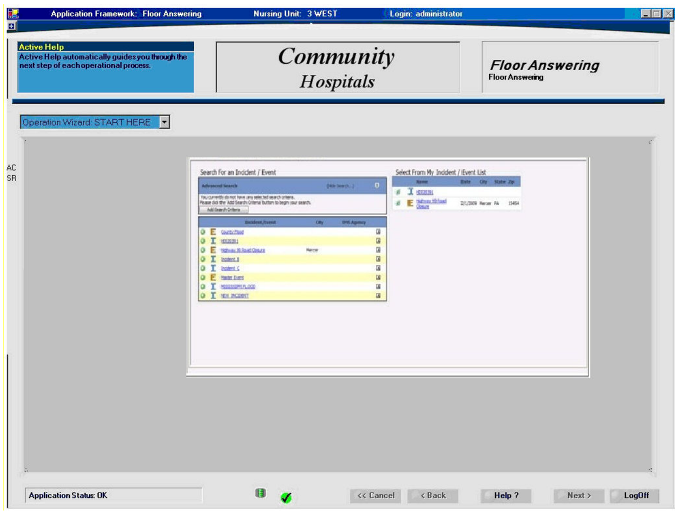
- Main menu
- Screen width (3D, full screen, and wide screen options)
- Header (active help, facility logo, and application name and type)
- Upper application buttons, lower application buttons, application dropdown list, application toolbar
- Upper operation buttons, lower operation buttons, operations dropdown list, operation toolbar

To simplify the element selections, 6 preset combinations, or themes, are easily presented to the user as follows:

2.1 Default Theme

Default theme is for the beginning single application user who has all options in a wizard, active help and no menu or multi-application options.

This option set shows a header (help, application name) for simplicity, limits options (no menu or application switching) and limits visual clutter (no buttons). All operations are in a drop-down list, with an integrated toolbar to encourage user growth as they become familiar with the application. 3D offset emphasizes application area (see Figure 1). Enabled feature options are Help, Facility Logo, Application Name, Operation Dropdown List, and Operation Toolbar.


Application Framework: Floor Answering   Nursing Unit: 3 WEST   Login: administrator

Active Help
Active Help automatically guides you through the next step of each operational process.

Community Hospitals

Floor Answering
Floor Answering

Operation Wizard: START HERE

Search For an Incident / Event

Select From My Incident / Event List

Application Status: OK

Figure 1: Default Theme

2.2 Demo Theme

Demo theme is for a multi-application user with touch screen that has the power (launch applications, touch buttons, logo), and simplicity (operation wizard, active help).

This option set is designed to highlight features of the application suite, and attract the customer at first glance. Help highlights simplicity, operation buttons highlight simplicity and touch compatibility, application buttons and name highlight the suite, the menu allows for customization to match the audience, and the facility name personalizes the demo. Features that are added to Default Theme are Menu, Upper Operation Buttons, and Lower Application Buttons (see Figure 2).

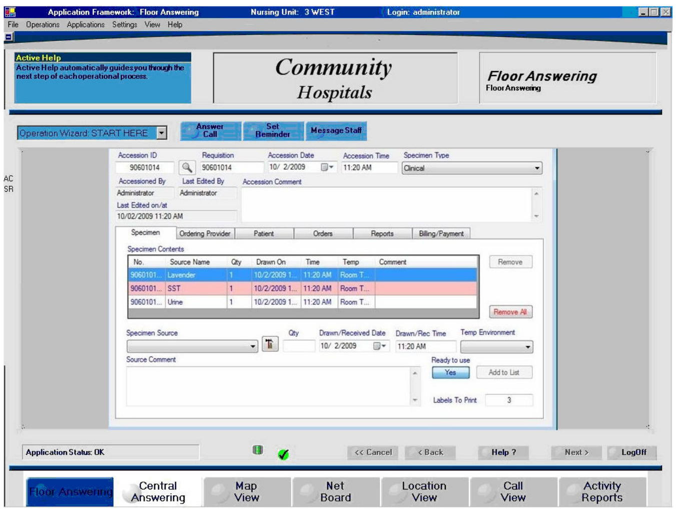


Figure 2: Demo Theme

2.3 Wide Theme

Wide theme is for the intermediate/expert user with touch screen that has upper application buttons, many operation buttons and its application screen is set to wide.

This option set is designed to provide maximum width, and more operation buttons, and switches the position of application and operation buttons. Features that are changed from the demo theme are, removed header (help, logo, application name), application buttons are moved to top, operation list and toolbar are removed, widened application area of screen, and operation buttons moved to bottom (button capacity grows from 5 to 9) (see Figure 3).

2.4 High-Low Theme

High-low theme is for the intermediate/expert touch screen user who has top-down flow using upper operation buttons and lower application buttons. This option set is designed to provide users familiar with menu operation as clean a screen as possible while still supporting touch screen for frequent operation and application switching. Putting the frequent operation buttons at the top of the screen supports top-down workflow. Intermediate and expert users gain some application screen space by eliminating the 3D effect. (Separation of frame and application is helpful to beginners). Features that are changed from the wide theme are operation buttons moved to top, application width set to full (no 3D), and application buttons moved to bottom (see Figure 4).

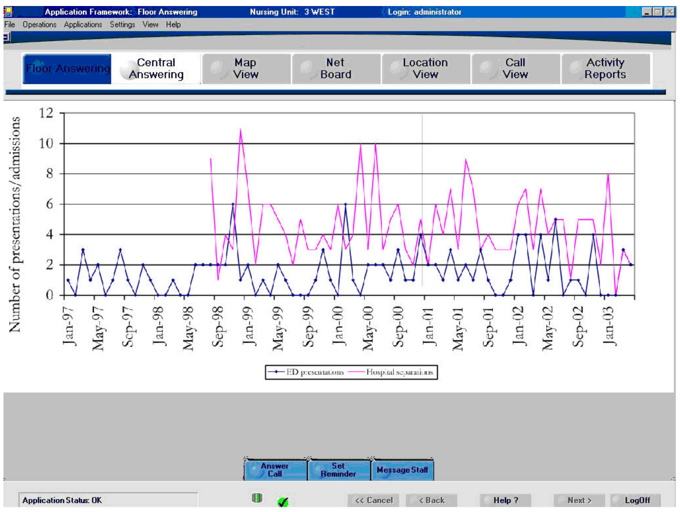


Figure 3: Wide Theme

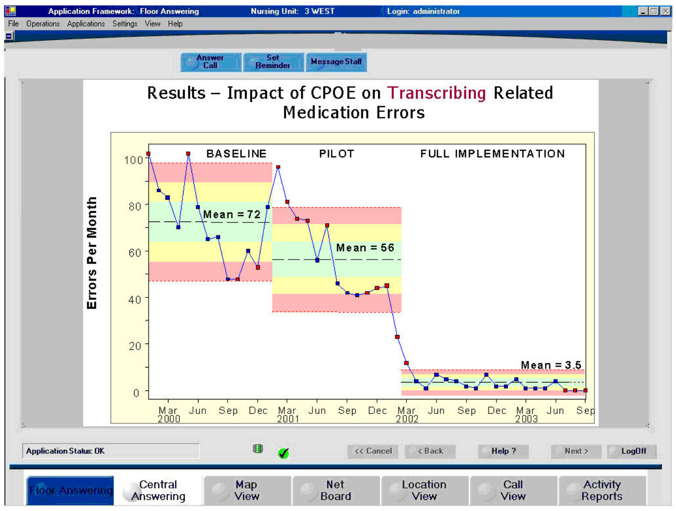


Figure 4: High-Low Theme

2.5 Plain Theme

Plain theme is for the intermediate/expert touch screen user who has menu-smart, upper buttons for known frequent operations, menu for other operations and applications. This option set is designed to provide menu-smart users more screen simplicity and application height by forcing application switching to a 2 step menu operation, but leaving the more frequently used operations as a single step touch screen operation. A feature that is changed from High-Low Theme is the Lower Application Buttons removed (see Figure 5).

2.6 Tall Theme

Tall theme is for the expert user without touch screen needing maximum rows with left-right flow using left side operation toolbar, right side applications toolbar and

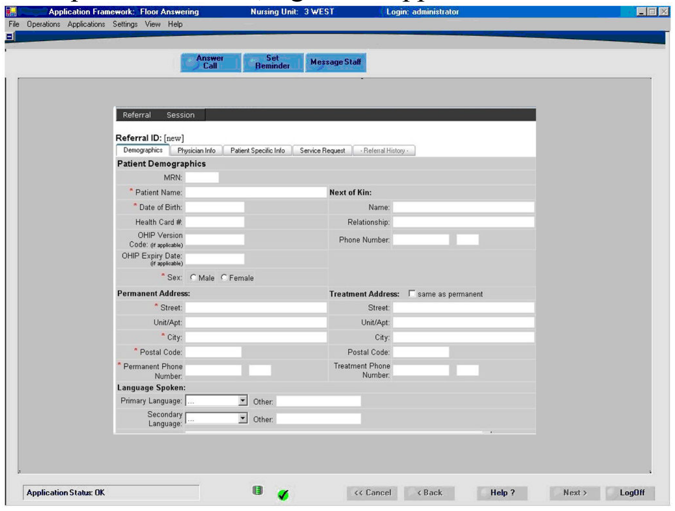


Figure 5: Plain Theme

no menu. This option set is designed to provide expert users knowing exactly what they want to do with maximum screen height and minimum frame clutter. It is not touch screen compatible, or wide-screen compatible. Features that are changed from the plain theme are menu removed, upper operation buttons removed, left side operation toolbar added, and right side application toolbar added (see Figure 6). The next section details the architecture for the applications and the host frame.

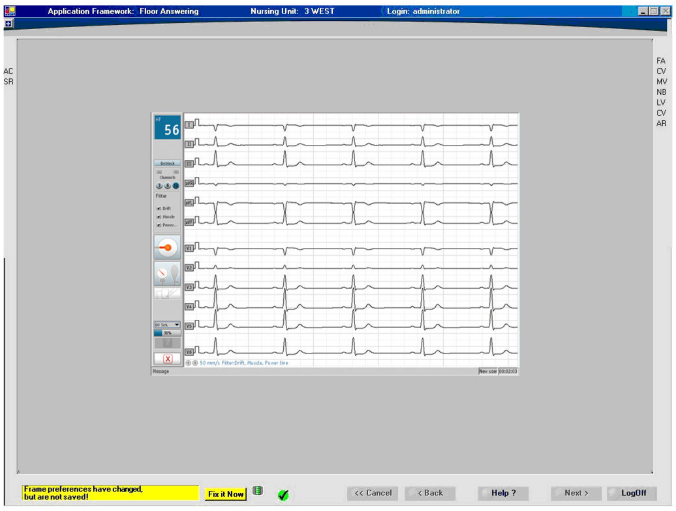


Figure 6: Tall Theme

3 Architecture

Each application that is loaded inside the host frame is a VB.NET DLL that implements the ICommunicate-WithApp interface (see Figure 7).

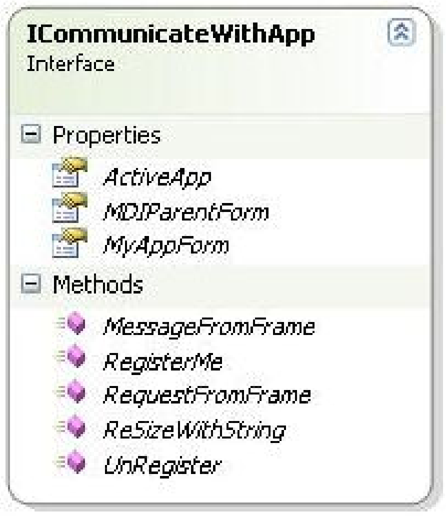
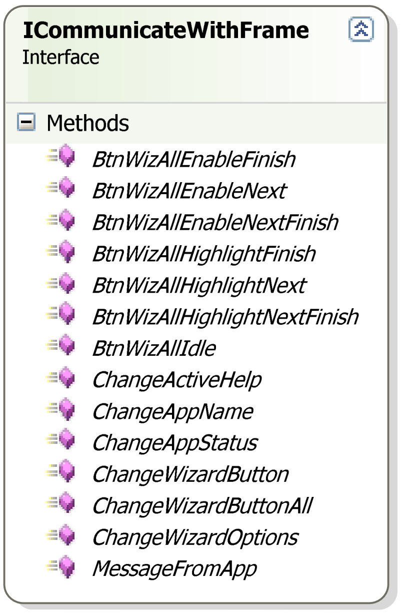


Figure 7: ICommunicateWithApp Interface

When the application is loaded but it is not in the foreground then the value for the *ActiveApp* is false so messages from user such as mouse or keyboard messages are not delivered to the background application by the host frame. Every application window DLL has its own main form that is displayed when the user brings it to the foreground. *MyAppForm* represents the main form of the application's window DLL. Host frame draws this *MyAppForm* inside frame's client area when *ActiveApp* for this application becomes *true*. *ResizeWithString* is the command sent from frame to application so that application adjusts its main form inside the frame's client area. *RegisterMe* passes the *ICommunicateWithFrame* interface (see Figure 8) to application so that application can send messages to host frame. These messages could be changes to the client area of the host frame such as displaying error or warning messages for the application. *UnRegister* is used when the *ActiveApp* becomes *false* and signals to the frame that the current application is about to switch to another application's main form. *MDIParentForm* is used by the application to add its main form as a MDI child to the frame. When the user makes changes on the screen and its *ActiveApp* is *true,* messages are sent to the application through *MessageFromFrame* command. When there is a request from frame and host frame expects the response from application in the same function call like login privileges, then frame uses *RequestFromFrame* command.

Host frame that is a VB.NET application implements the *ICommunicateWithFrame* interface to communicate with the foreground application (see Figure 8).



ICommunicateWithFrame
Interface

Methods

BtnWizAllEnableFinish
BtnWizAllEnableNext
BtnWizAllEnableNextFinish
BtnWizAllHighlightFinish
BtnWizAllHighlightNext
BtnWizAllHighlightNextFinish
BtnWizAllIdle
ChangeActiveHelp
ChangeAppName
ChangeAppStatus
ChangeWizardButton
ChangeWizardButtonAll
ChangeWizardOptions
MessageFromApp

Figure 8: ICommunicateWithFrame Interface

Each application sends messages to host frame using *MessageFromApp* command. These messages are like *Logoff* and so on. Furthermore, *BtnWizAll** commands are used by the foreground application to enable/disable buttons such as *Next, Finish, Cancel,* etc. during user's wizard operations. Foreground application can change frame's caption, company's logo, text content for help, and menu items using *Change** commands.

On the right, you may see the code snippets for starting an application and logoff process.

4 Status Line with a Correction Button

It is common for applications to employ a status line (see Figure 9) to alert the user of the condition of the application. Specifically, the status line contents can include passive messages, such as reassurance of an acceptable condition or notification of an application being too busy to accept user input. The status line can also contain an active message requiring user action, such as a prompt for the most likely user choices or a warning that some settings within the application are invalid and should be corrected. However, this is where the usefulness of the status line ends. It is typically up to the user to determine how to perform the action suggested on the status line. A correction button that performs a simple suggested action (with no options), or takes the user directly to a screen to do so for more complex suggested actions (with several options) greatly enhances the usefulness of the status line and the simplicity of the application [3, 4, 5, 6].

```
Public Sub StartApp(ByVal sDirectory As String, ByVal
sTypeName As String,
    ByRef MyAssembly As System.Reflection.Assembly, _
    ByRef MyApp As ICommunicateWithApp, ByRef
MyCurrentApp As CApp, _
    ByVal sMenuText As String)
    Dim A As New CApp
    A.MyAssembly = A.MyAssembly.LoadFrom(sDirectory)
 'create the interface, so frame can send messages to the
application
    A.MyApp = CType(A.MyAssembly.CreateInstance
(sTypeName), _
    ICommunicateWithApp)
    A.MyApp.ActiveApp = True
' create the reference inside the application so app can talk
' to the frame
    A.MyApp.RegisterMe(boss)
    A.MyApp.MDIParentForm = boss
' add the CApp.MyApp.MyForm into the panel, so it can be
drawn in the client ' area
    boss.pnlWideScreen.Controls.Add(A.MyApp.MyAppForm)
    MyCurrentApp = A
    A.MyApp.MessageFromFrame("<Login><UserName>" &
boss.UserName & "</UserName><Passwd>" & _
    boss.UserPass & "</Passwd><DB>" & DBServer &
"</DB></Login>", Nothing, "Login")
    A.MyAppName = sMenuText
    A.MyFileAppName = sMenuText
    Me.AddApp(A)

A.MyApp.ReSizeWithString("<EOF><TOP>0</TOP><LEFT>
" & _
    "0</LEFT><HEIGHT>" &
CStr(boss.pnlWideScreen.Height) & "</HEIGHT><WIDTH>"
& _
    CStr(boss.pnlWideScreen.Width) & "</WIDTH></EOF>")
End Sub
```

Code Snippet for Starting an Application

```
Public Sub LogOff(ByVal sName As String, Optional ByVal
bToSwitch As Boolean = True)
    Dim a As CApp
    Dim i As Integer = 0
    For Each a In Me
      If a.MyAppName = sName Then
         boss.pnlWideScreen.Controls.Remove
             (a.MyApp.MyAppForm)
         boss.MyApp = Nothing
         boss.MyAssembly = Nothing
         ' another ref for current CApp object
         boss.MyCurrentApp = Nothing
         UnRegister(a.MyApp)
         a.MyAssembly = Nothing
      end if
    next
end sub
```

Code Snippet for Logoff

The next section shows how to improve a typical status line in the host frame.

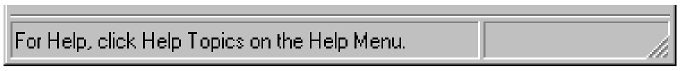

For Help, click Help Topics on the Help Menu.

Figure 9: Typical Status Line

5 Enhanced Status Line

This paper introduces an enhanced status line with the typical text property, but with the added properties of text and background color, whether the button is visible, and the button text. As with typical status lines today, an application determines at any given time the status text to display. In addition, the application determines a background color based on how critical the status message is. Simple assurances that everything is OK typically have a background color that matches the surrounding area, so that the status message does not distract the user. More urgent messages have a contrasting color designed to be noticeable, while the most urgent messages use a bright color specifically designated to attract the user's attention. The text color in all cases should be set to be readable against the designated background color [7, 8, and 9].

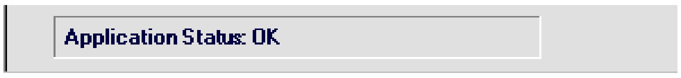

Application Status: OK

Figure 10: Enhanced Status Line Showing a Passive Message

If the status message indicates a condition that should be corrected, the designer sets the "button is visible" property to true, and sets the button text. Text and background colors of the button match those of the status message. When the user selects the button, a Status Correction event is sent to the application, which responds appropriately to correct the faulty status immediately or jump to a screen in the application that queries the user for needed information before selecting the appropriate status correction action.

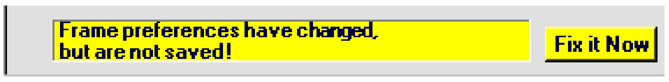

Frame preferences have changed, but are not saved!

Figure 11: Enhanced Status Line Showing a Suggested Action and Correction Button

5.1 Enhanced Status Line Design Architecture

Figure 12 shows the design of the status display. The application determines its own status based on its current state, and passes that to the application status controller. This controller converts the status into the five status display properties, via a lookup table. The table stores all 5 values for all possible status conditions.

The status controller sets the status property values. These properties are dynamic, and can be set by the status controller upon a value change only, as a periodic background task for maintenance purposes, or as a combination of both, depending on the response time needed by the application. Additional static status display properties, such as position, size, font, etc. of the status line and button are set at design time and remain fixed throughout the life of the application.

The text and color drawing controller periodically polls the status display properties, and creates a drawing based on them. The drawing is fed to the IO device, typically a VGA terminal or equivalent.

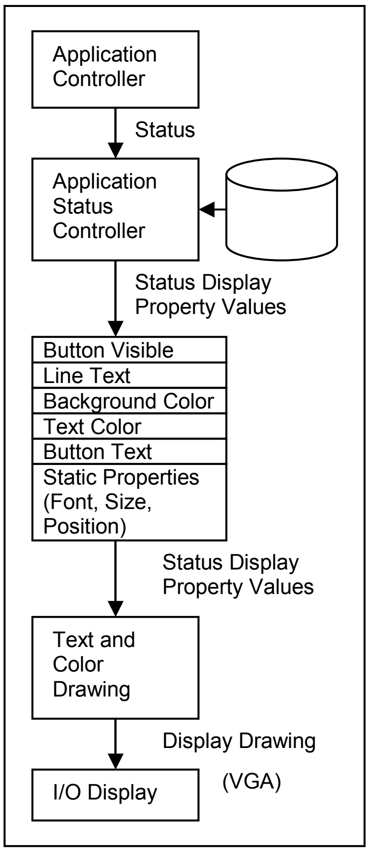


Figure 12: Status Display Design Flow

Figure 13 shows the design flow for a user request for status correction. The IO sends a mouse click event to the Input Controller, which determines that the click occurred within the confines of the Status Correction Button. This event is raised to the Status Controller, which converts it to a Status Correction event which is passed on to the application's Status Controller. The application knows what the current status is, and the action needed to correct the status. If it has all the information needed, it will immediately take corrective action on the faulty condition. If, however, the corrective action is complex enough that additional user input is necessary, a user input screen is displayed, describing the condition, potential solutions, and how the user selections will affect the choice of solutions. Once all required user input is obtained, the corrective action can be taken.

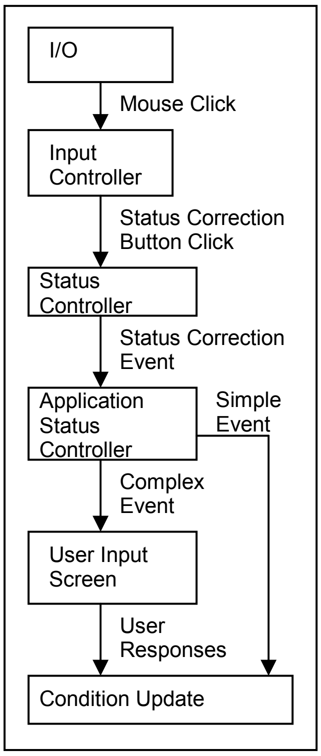


Figure 13: Status Correction Design

While many applications use a status line to alert users of acceptable or unacceptable state of the application, none allow for integral correction of an unacceptable condition within the status line itself. This paper incorporates this new feature into host frame design, simplifying the operation of each of these applications by applying the basic user interface design principle of "combine display and update capabilities" to the status line.

6 Conclusion

In this paper, host frame and enhanced status line concepts are developed. Common elements that are depicted in Table 1 are drawn in the periphery of the host frame's client area. Host frame implements the ICommunicateWithFrame software interface (see Figure 8). Each application is successfully loaded inside host frame's client area as long as the interface class (ICommunicateWithApp) that is given in Figure 7 is implemented. Each foreground application can change the frame's caption; help content, and menu items. The communication between the frame and this foreground application is carried out through interface classes given in Figure 7 and 8.

It is common for applications to employ a status line to alert the user of the condition of the application. Specifically, the status line contents can include passive messages, such as reassurance of an acceptable condition or notification of an application being too busy to accept user input. The status line can also contain an active message requiring user action, such as a prompt for the most likely user choices or a warning that some settings within the application are invalid and should be corrected. However, this is where the usefulness of the status line ends. It is typically up to the user to determine how to perform the action suggested on the status line. An integral button that performs a simple (with no options) suggested action, or takes the user directly to a screen to do so for more complexes (with several options) suggest actions greatly enhances the usefulness of the status line and the simplicity of the application.

This paper introduces an enhanced status line with the typical text property, but with the added properties of status level, back color, button visible, and button text. When the button is selected by the user, a Status Correction event will occur, allowing the application both to control the look and feel of the advanced status line / button combination and to respond appropriately to user requests to correct a status condition requiring action.

References

[1] G. D. Abowd and R. Beale, Users, Systems and Interfaces: A Unifying Framework for Interaction, *People and Computers VI*, pp. 73-87, Cambridge University Press, Cambridge, 1991.
[2] L. Bass and J. Coutaz, *Developing Software for the User Interface*, Addison-Wesley, New York, 1991.
[3] M. Beigel, H. W. Gellersen and A. Schmidt, MediaCups: Experience with Design and Use of Computer-augmented Everyday Objects, Computer Networks, pp. 35(4):401-9, Special Issue on Pervasive Computing, Elsevier, March 2001.
[4] B. W. Boehm, Verifying and Validating Software Requirements and Design Specifications, *IEEE Software*, January 1984, pp. 75-88.
[5] J. Coutaz, Architectural Design for User Interfaces, *Proceedings of the 3rd European Conference of Software Engineering*, ESEC'91, 1991.
[6] C. Gram and G. Cocton, editors, *Design Principles for Interactive Software*, Chapman and Hall, London, 1996.
[7] A. Howes and S. Payne, Display-based Competence: Towards User Models for Menu-driven Interfaces.

International Journal. of Man-Machine Studies, pp. 33:637-55, 1990.
[8] D. J. Mayhew, *Principles and Guidelines in Software and User Interface Design*, Prentice Hall, Englewood Cliffs, NJ, 1992.
[9] A. Monk, editor, *Fundamentals of Human-Computer Interaction*. Academic Press, London, 1985.
[10] B. A. Myers, *Creating User Interfaces by Demonstration*. Academic Press, New York, 1988.
[11] B. A. Myers and M. B. Rosson, Survey on User Interface Programming, *Conference Proceedings on Human Factors in Computing Systems*, pp. 195-202, ACM Press, New York, 1992.
[12] I. Sommerville, *Software Engineering*, 4th edition, Addison-Wesley, Workingham, 1992.