Position paper and UI design pattern included

Position paper (really, just who I am and my interest in patterns):

I have an established interest in, and participation in the dissemination of, design patterns: I am coauthor of *The Design Patterns Smalltalk Companion* (Alpert, Brown, & Woolf; 1998). This is a follow-on volume to the Gang of Four’s *Design Patterns* book (Gamma et al., 1995). We (the authors) use their 23 software design patterns as a starting point and view them through Smalltalk glasses (explain and exemplify them in the context of the Smalltalk programming language) while elaborating on and adding to their pattern inventory. I worked closely with the Gang of Four while writing our book.

I’ve also worked and published in the fields of HCI, software engineering, multimedia, and instructional technology at IBM’s T.J. Watson Research Center since 1987. I’m interested in blending my experience in design patterns with my interest in HCI and I feel that a solid inventory of UI/HCI patterns would be an extremely valuable tool to interface designers, just as software patterns can help software designers.

I’ve read Alexander’s *A Timeless way of Building* and Alexander et al.’s *A Pattern Language* (and quoted/cited both in my book).

I’ve attached a sample UI pattern, *Sharing Screen Space Among Multiple Application Contexts, or “Real Estate is Expensive.”*

References

Sharing Screen Space Among Multiple Application Contexts, or “Real Estate is Expensive”

Sherman R. Alpert
IBM T.J. Watson Research Center
alpert@watson.ibm.com

Intent

Provide a mechanism for sharing screen space/real estate such that the same real estate is reused for different UI elements when there is a change in application state, context, or mode, or user task or needs.

Motivation

Many applications have a lot of information to offer for user consumption. It’s often difficult to find the space to provide all such information on the screen simultaneously – screen real estate is expensive. Just as importantly, it is also common that only a specific subset of such information or UI widgets is relevant in a particular application context or state, or for a particular user task, but is not germane in others. That is, different sets of information/data/UI widgets may be relevant in mutually exclusive contexts. This is a core criterion of this pattern. If the application just has a lot of information to show in all contexts, the information can simply be displayed in a scrollable pane – hence, though the information will not fit within a rectangular space available on the screen, the user has easy access to all of it by scrolling. Nonetheless, even this may be unsatisfactory: the total information available to the user, or the entire set of UI widgets available for all tasks, may be classifiable into different subsets depending on the user’s current informational needs or task, and showing all possible information at once makes it difficult for the user to find just what she is looking for.

Thus, the application scenario addressed by this pattern entails: when the context, state, or user’s informational needs or task changes, a new set of information or a new UI-subset must be displayed and the information that was visible during the previous context is not now relevant and should no longer be visible. Again, if the application simultaneously displays all the information and interactive UI elements for all contexts, this may easily confuse or overwhelm the user (“Now, where – among this morass of widgets – are the widgets I need to look at and interact with for my current task?”). So, by selectively showing only those UI elements that are relevant for the current task, we support the user in accomplishing that task.
In some cases, the full set of widgets in the entire window must change for each context, state, user task, or user informational need (Let’s simply call these context from here on). In other situations, there may be a subset of information and widgets that are common to, and required for, all sub-contexts, and only a portion of the screen must change depending on context. In either case, we might label the problem “the context-dependent UI problem.”

Here’s a simple example. In the Microsoft Windows™ Control Panel, the user may select the sub-control-panel for multimedia, the Multimedia Properties panel. Here, the user may modify settings for different multimedia devices. Since audio devices and video devices (etc.) are separate contexts, the information and widgets allowing users to learn about and set control values for each of these devices need not be shown simultaneously. Indeed, they should not be shown simultaneously for fear of confusing the user – settings for audio are mutually exclusive with those for video – and because together they would require a huge window. The Multimedia Properties panel’s solution is to use a multi-tab control, with a tab for each type of device (Figure 1).

![Multimedia Properties panel](image)

Figure 1. The Windows Multimedia Properties control panel, with a tabbed control allowing separate information and widgets to be displayed, and interacted with, for each type of multimedia device.
Applicability

This pattern applies when there is too much information or too many widgets to fit within an application’s window and/or some information and widgets are applicable/relevant to Context A, while other data and widgets to Context B, and so on. That is:

1. A single application has multiple contexts, states, tasks, or user informational needs.
2. Each of these contexts requires the user to interact with UI elements that are relevant to that context alone, but:
3. We don’t want to overwhelm or confuse the user with a window that simultaneously displays all UI elements relevant to all contexts, including those not relevant to the application’s or user’s current context.
4. Hence, some or all of the UI elements in the application’s window should change dependent on context.

Implementation

There are a number of ways to provide a solution to the real estate problem described above, all centering around the notion of reusing a single portion of the screen or window for different purposes and in different contexts.

Returning to our Multimedia Properties control panel example, one solution to reusing a single portion of a window for different contexts is a tabbed control (I’m using control here as a synonym for widget). Some tabbed controls are made to appear on the screen as a notebook, for example with a spiral binder on the side opposite the tabs; this is called a notebook control or tabbed notebook control.

Another “solution” occurs in the Multimedia Properties control panel, as well. When the “Devices” tab is selected by the user, the UI appears as in Figure 2.
If the user wishes to gain access to the properties for the selected device (to view or modify them), the Multimedia Properties panel no longer reuses part of its own window, but rather pops up a separate window (see Figure 3). This is another solution to the context-dependent UI problem. This solution does not reuse part of the application’s window (although it may reuse a portion of the screen if it opens up directly on top of the existing Multimedia Properties panel and is not larger than the latter). Nonetheless, this solution has the significant drawback of making the user now manage multiple windows for a single task.
Figure 3. When the user tries to gain access to the properties for a specific device, a separate window – different for each device – is opened.

Another solution to the context-dependent UI problem is to use radio buttons to “flip” among different views that share the same window area. When the context changes, the application displays different information in the same screen area. The user may choose to view alternative information in that screen area by selecting one of the radio buttons – the set of information and UI elements displayed in the shared portion of the window is dependent on which button is currently selected. Also, the application itself may programmatically change the information context by “selecting” one of the radio buttons. Let’s look at a concrete example.

In EFX, a digital-video/film/audio editor, there is a significant amount of information to always be shown and some information that is dependent on the current user task or context or information needs. Here, especially because a large portion of the screen had to remain “constant” – showing the same widgets in all contexts – we have the classic real estate problem. The solution was to have the top of the screen always display the same information, and make the bottom portion “multipurpose.” The information shown in this latter section was controlled by the selection of one of several radio buttons (This was normally
under user control, but the application could also programmatically “select” a radio button thereby changing the contents of the “multipurpose pane”.

In Figure 4, (it’s a bit difficult to read, but) below the timeline are radio buttons (on the left) that control what is displayed in the “multipurpose pane” (on the right). Note that the pattern is applied recursively in this application. When the “Prop’s” (“properties” – again a real estate problem) button is selected, the Property Sheet for the currently selected timeline element is displayed in the multipurpose pane. Since there are many properties and many of these require a separate set of widgets for setting the individual property, the Property Sheet is divided into a list of properties (on the left) and the UI elements required to set an individual property’s values (on the right). A specific property can be selected via its radio button, resulting in the display of the UI for setting that property (see Figure 4).

![Figure 4. The EFX UI. On the bottom left are a set of radio buttons that govern what is displayed in the multipurpose pane on the lower right. The “Prop’s” button is selected to display the Property Sheet. In the Property Sheet, the “Percent” property’s radio button is selected, causing the display of the widgets to set this property’s value. Thus, the “Sharing Screen Space Among Multiple Application Contexts” pattern is applied recursively.](image)

Note that the “radio button solution” need not use “standard” radio buttons. Figure 5 shows the UI of a tutor for algebra equation solving. The right side of the screen is a “blackboard” that can display different information and UI elements depending on the selection of radio buttons below the blackboard (labeled “Advice” for obtaining help from the application, “History” for displaying a history of user problem-solving interactions, and “Graph” for viewing an interactive graph of the current equation). The point here is that the radio buttons...
must merely *behave* as mutually exclusive selectors controlling what is displayed in the shared window area, regardless of their appearance.

![Figure 5](image)

Figure 5. Controlling what appears in a multipurpose pane using the “radio buttons solution” requires that selector buttons *behave* as radio buttons; their appearance may vary from “standard” radio buttons.

This pattern is often applied on Web sites as well. By using HTML frames, a site can maintain constant (always visible) information and UI elements in one frame and changing (depending on context) elements in a separate frame. This has been applied on Web sites in a number of ways. I mention here a common implementation. As exemplified in Figure 6, many Web sites have a constantly visible navigation bar in one frame (the left frame in the figure). A user may click any of the hyperlinks in the navigation bar to change to contents of the “content frame” on the right. We thus have a Web-based application of the “Sharing Screen Space Among Multiple Application Contexts” pattern.
Consequences

Upside:
- Less confusion for user – user may not know where to find the information/widgets relevant to his/her task.
- Support users in accomplishing tasks.

Downside:
- More coding effort.
- Possibly could still be confusing for user: Where is the page/information/UI relevant to the user’s task?

Known uses
- Windows Control Panel and sub-windows use the multiple tab control solution.
- AlgeBrain, an intelligent tutoring system for algebraic equation solving (Alpert, Singley, & Fairweather, 1999) uses the radio-buttons-control-multipurpose-pane solution.
- EFX (Alpert et al., 1995) uses the radio button plus multipurpose pane solution recursively.
- The OOPSLA 2000 Web site (OOPSLA, 2000) uses a persistently visible frame that contains hyperlinks which control the contents of a changeable content pane.

References
