Paper 10-27

Designing Web Applications: Lessons from SAS® User Interface Analysts

Todd Barlow, SAS Institute Inc., Cary, NC

ABSTRACT

Web application user interfaces combine aspects of non-Web GUI design and Web site design. Through iterative usability testing, SAS user interface analysts have developed standard design and interaction patterns for analytical Web applications.

INTRODUCTION

Web application user interfaces differ from those of traditional Windows or Java applications. The degree of interactivity possible in a Windows or Java application is difficult to achieve in a Web application. Web applications differ from Web sites too. Compared to most Web sites, Web applications are targeted at a more specific audience with a limited but complex set of goals. To date, our Web applications have been tools for performing a variety of analytical tasks. Most allow the user to choose a type of analysis, provide parameters for the analysis, and obtain the results of the analysis.

Over the past year, SAS user interface analysts have been designing and testing user interfaces for Web applications. Part of the design effort was the development of the SAS Web GUI standards. These guidelines address the appearance of Web applications and suggest high-level schemes for page layout and navigation in the application. Usability testing of Web applications has confirmed that usable Web applications combine design practices of traditional Windows GUI design and Web site design. This paper describes what we have found during the past year.

NAVIGATION STRUCTURE AND LAYOUT

A survey of popular Web sites shows that site designers have converged on a limited set of common page layout and navigational schemes. The SAS Web GUI standards build on those common schemes to take advantage of users' familiarity with them. The following section describes one of the schemes that is well suited for analytical Web applications. Figure 1 is an example of the GUI for this scheme. The page contains four parts: a banner, a tabbed navigation area, a vertical menu, and a content area. Figure 2 shows the organization of pages in the application for this scheme. It is a hierarchical organization with the application's home page at the top of the hierarchy. The remaining pages are grouped functionally. Within those groups, smaller groups of pages may also be hierarchically grouped.

LAYOUT STRUCTURE

The banner contains the application name and the SAS logo. The application name is a link. Clicking on it loads the application's home page. (The SAS logo is not a link.)

The banner can also contain breadcrumbs. Figure 3 gives an example of breadcrumbs, the series of links that show the position of the page in the hierarchy of pages in the application. Breadcrumbs show the path navigated to arrive at the current page. The purpose of showing the path is to expose the structure of the application and help the user learn the location of pages in the application.

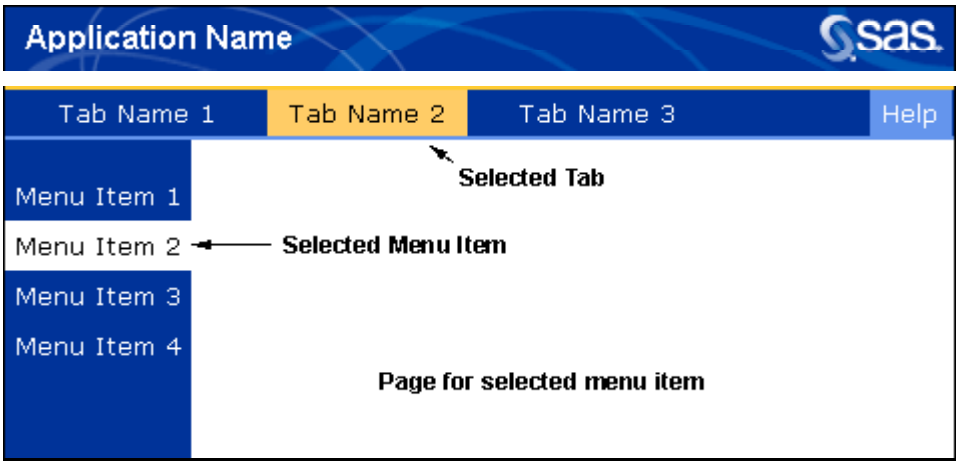

Figure 1. SAS Web GUI Standard Page Layout

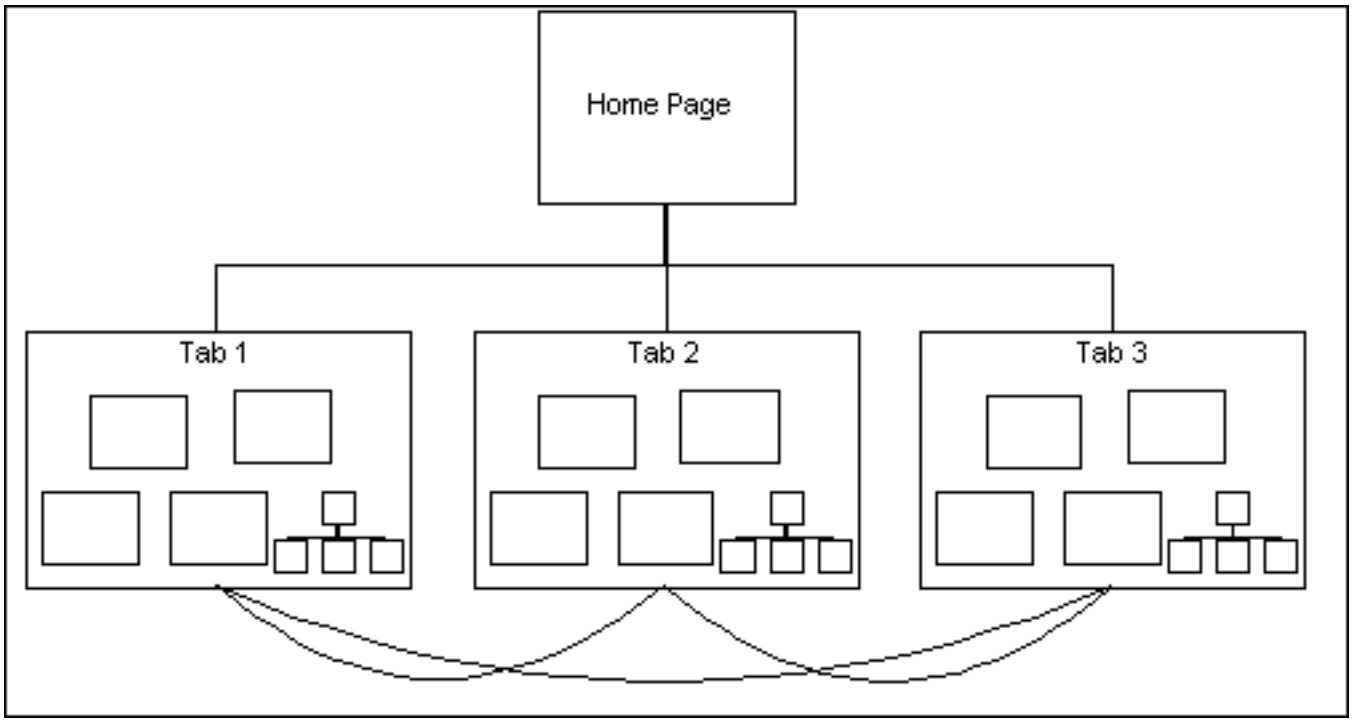

Figure 2. Page and Link Structure for Web Application

Figure 3. Breadcrumbs in the Banner

It is also a navigational tool for situations in which the user wants to return to an earlier page in the sequence. Breadcrumbs can be useful if the site is hierarchical and does not have too many levels. However, it is not clear if breadcrumbs are really worthwhile. In our usability tests, almost nobody used them even after we pointed them out. Some users told us that they didn't expect any functionality in that area of the screen. They assumed that the application name and the breadcrumb text were for informational purposes only.

The area beneath the banner contains tabs that describe broad areas of the application that, when clicked, open the main page for that area. In a traditional Windows GUI, tabs are used to separate groups of controls that are somehow related but do not interact with each other. There should not be any dependencies among the controls on different tabs. Nor should it be possible to move from one tab from another except by clicking on the tabbed part of the window. In a Web application, tabs also group similar sets of features together. Unlike a traditional Windows GUI, features grouped under one tab can be (and should be) accessible through pages accessed from another tab. Moving from tab to tab without an explicit click on the tab is fine — as long as the movement is meaningful within the context of the task and can be reversed. The curved lines near the bottom of Figure 2 represent links between pages in different functional groups. Problems arise when the user is unable to return to a page within a tab without clicking on the tab. In other words, you should not have a one-way path from one tab's area to another tab's area. If the user needs to return to an earlier page, the user will look for a similar navigational path. Users will only try the tabs after exhausting other options, probably not noticing the subtle change in tab color that happened when navigating between the tab's areas.

The rectangular area on the left side of the page beneath the tabs is a vertical menu. It controls navigation within the area defined by the selected tab or controls navigation on a single page. When it controls navigation within the tab's area, clicking on a menu item loads a new page on the right side of the window. The smaller rectangles in Figure 1 represent the pages associated with the menu items in the vertical menu. The menu can have more than one

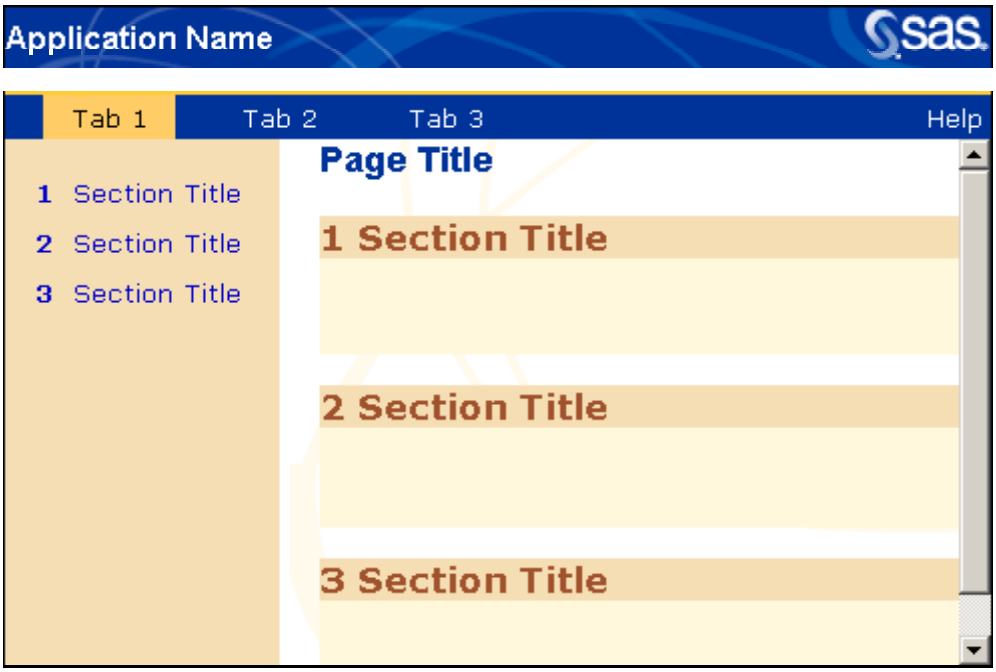

Figure 4. Multiple Steps on One Page

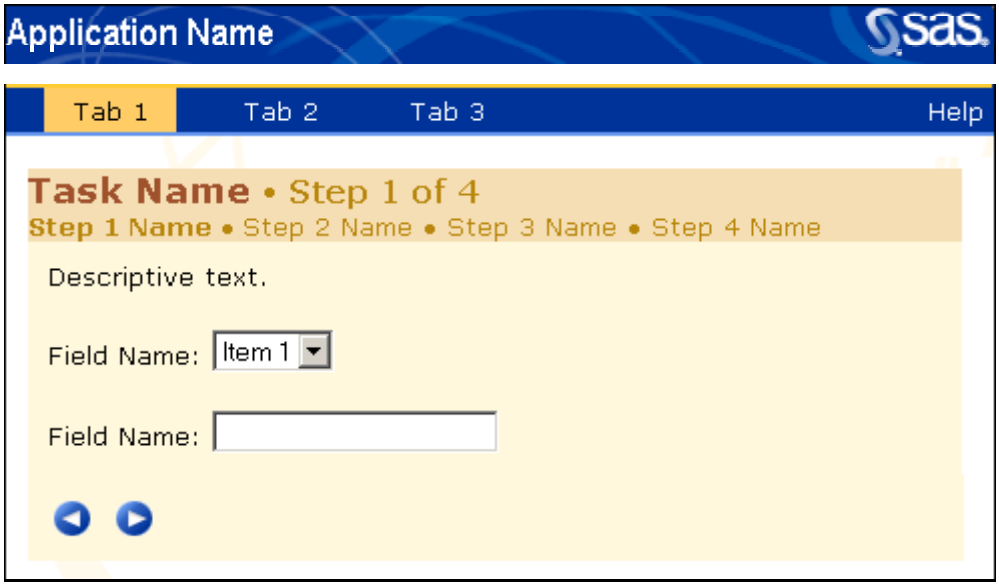

Figure 5. Multiple Steps Across Several Pages

level to enable the pages within a functional group to be hierarchically organized. When the menu controls navigation in a page, clicking on a link scrolls to a bookmark in the page. Figure 4 illustrates this type of page.

Our first designs used the same color scheme in both situations. Testing suggested that the color scheme, which was dominated by the same blue used in the banner and tabs, was a problem when the links scrolled to bookmarks on the page. Rarely did people look at that area as a way to navigate within the page. The similarity in color made the area appear as part of a higher-level navigational tool. As a result, we changed the color of the left side to match the color scheme within the page in an attempt to connect the two visually and suggest a functional relationship. (See the on-line version of the paper to see the color figures.)

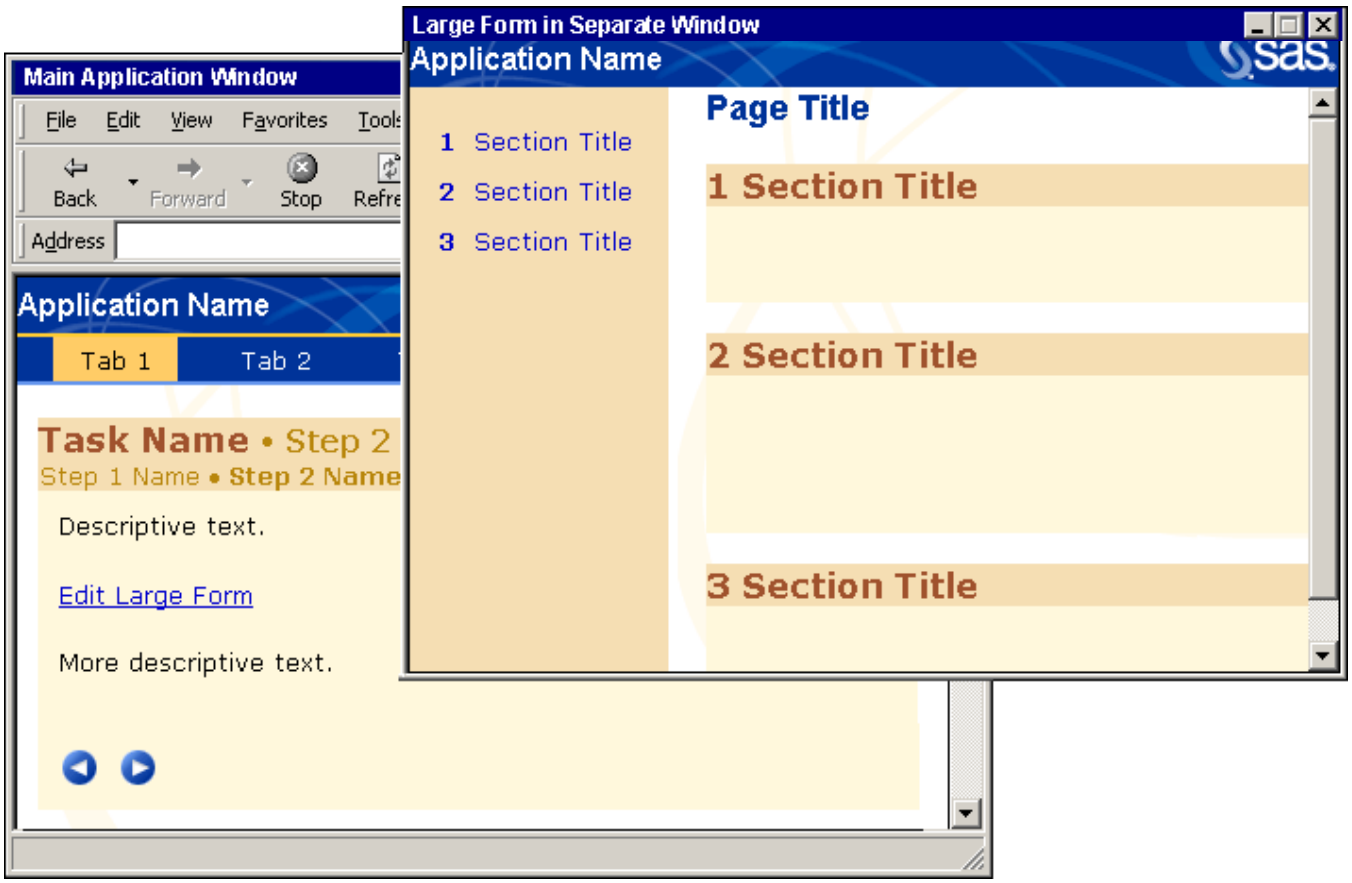

Figure 6. Multiple Browser Windows

PAGES AND NAVIGATION

The decision as to what to include on a page and the order in which pages appear depends on a task analysis of the user's goals. We design pages that capture only the information needed to accomplish meaningful components of the task. The balance between too many subtasks and too few subtasks on a page is achieved through understanding how the user makes decisions while performing the task and assigning functions to the system and to the user. The result of this approach is a structured way of accomplishing goals through a predetermined set of steps that help the user reach the goal. The important parts of the user interface are made salient at the appropriate times. This approach also makes it easier for users to form strategies for chaining multiple tasks to accomplish more complex goals that may not be represented explicitly in the user interface.

At the task level, there are two ways we present structured tasks. When the task has multiple steps that can be completed in any order because there are no interactions between them, then we use a single page with numbered subheadings and a navigational area on the left. The subheadings define areas within the page that contain controls associated with subtask completion. Figure 4 illustrates this type of page. These pages contain at least two command buttons: **Cancel**, which returns the user to the previous page and does not save any of the work done on this page, and a button that saves the work and acts upon the contents of the page, e.g., **OK**.

When the task has multiple steps and the completion of one step depends on the successful completion of a previous step, then we design separate pages and lead the user through them. Figure 5 shows an example of this type of page. These pages have much in common with wizards in a traditional Windows GUI.

We have added a few features of our own:

- Each page shows the name of every page in the sequence so that users will know what to expect and when to expect it.
- The browser's history feature enables the user to move more than one step backward in the sequence. Moving more than one step is only possible when moving backward. The application does not enable moving more than one step forward in the sequence.

- The pages are not modal. At any point in the sequence, the user can navigate to another part of the application. Leaving the sequence cancels any work in progress.

MULTIPLE BROWSER WINDOWS

Opening a new browser window is necessary in some situations and advantageous in others. We open new browser windows in two situations. One is when we want to maintain the context of a task but need more information than can be accommodated on the current page. Figure 6 shows an example of this situation. The new browser window is modal because the window is needed only for the task being performed in the main browser window. The second situation is when the task analysis suggests that comparing the contents of two pages is necessary. Here, the second browser window is modeless. For both modal and modeless windows, the following guidelines apply:

- The application's primary window should close the other browser windows at the right times. For example, if the user closes the primary browser, then all other windows on the application should also close.
- New browser windows should not contain the typical browser controls found in the main application window, that is, **Back**, **Forward**, url field, and pull-down menus.
- In the main application window, the link that opens a new window should indicate that a new window will open.

PROPERTY SHEETS AS SEPARATE WINDOWS

We try not to re-create the ubiquitous “Properties” windows in traditional Windows GUIs. Usability testing shows that they do not work well as modeless windows. A similar problem has been observed with modeless Help windows. The problem we have observed is that people do not always remember to close them. When they need them again, they click on the control that should open the property page and nothing happens because the page is already open but has been covered by the main application window. Few users realize that the window is already open. Calling them back to the top of all windows can be difficult. The user could use the taskbar to see which windows are open, but that requires that the user understand that (1) the application created a separate browser window to use as a property page, (2) the window will show a separate button on the taskbar (which is unlike non-Web applications), and (3) their taskbar is not so crowded that they can find the right button. In most cases, we have solved the problem by putting the controls from the new browser window on a new page inside the main application window.

TABLE DESIGN

A common Web application control is a table containing a list of work objects (for example, analyses, reports, data sets). Each contains the name and other information about the object. The pages on which these tables appear provide basic maintenance features, such as **Copy**, **Delete**, and **Rename**, and features that are unique to the application. We compared two designs, shown in Figure 7, for this type of table. In the first design, each row contained buttons or links that acted on the object in the row. The second design had a single set of links for the actions and a check box on each row in the table. The advantage of the first design is that the action of the link is unambiguous. Its location on the same row as the object makes it apparent that it will act on the object in that row. The disadvantages are that the design does not scale well and prevents the user from acting on more than one object simultaneously. The advantage of the second design is that it solves the problems of the first approach. Unfortunately, it creates a new problem whose impact we are still trying to understand. Users insist on checking the check boxes even when the check box has no effect on their actions. In several usability tests, we have observed nearly every participant checking the check boxes when it was not necessary. The two designs are not mutually exclusive. Our table controls follow these guidelines:

- The name of the item should be a link that opens the item.
- Actions that can only be performed on one item at a time should be put on the same row as the item.
- Actions that can be performed on several items simultaneously should be put below the table and should apply to the items that are checked.

INTERACTIVITY AND RESPONSE TIME

Most people are not aware of the differences in system architecture between a Web application and a PC-based application. We frequently get comments from test participants about slow response times and the lack of information about what is happening. Their experience with Windows applications influences their expectations about interactivity in Web applications.

Given the increasingly large amount of data being manipulated through a Web application, it is not uncommon for some server-side processing to take more time than users are willing to wait. Response times can range from instantaneous to several minutes. We have come up with several approaches to dealing with lengthy delays in Web applications.

- Prevent the user from interacting with the application while the server is working. If the

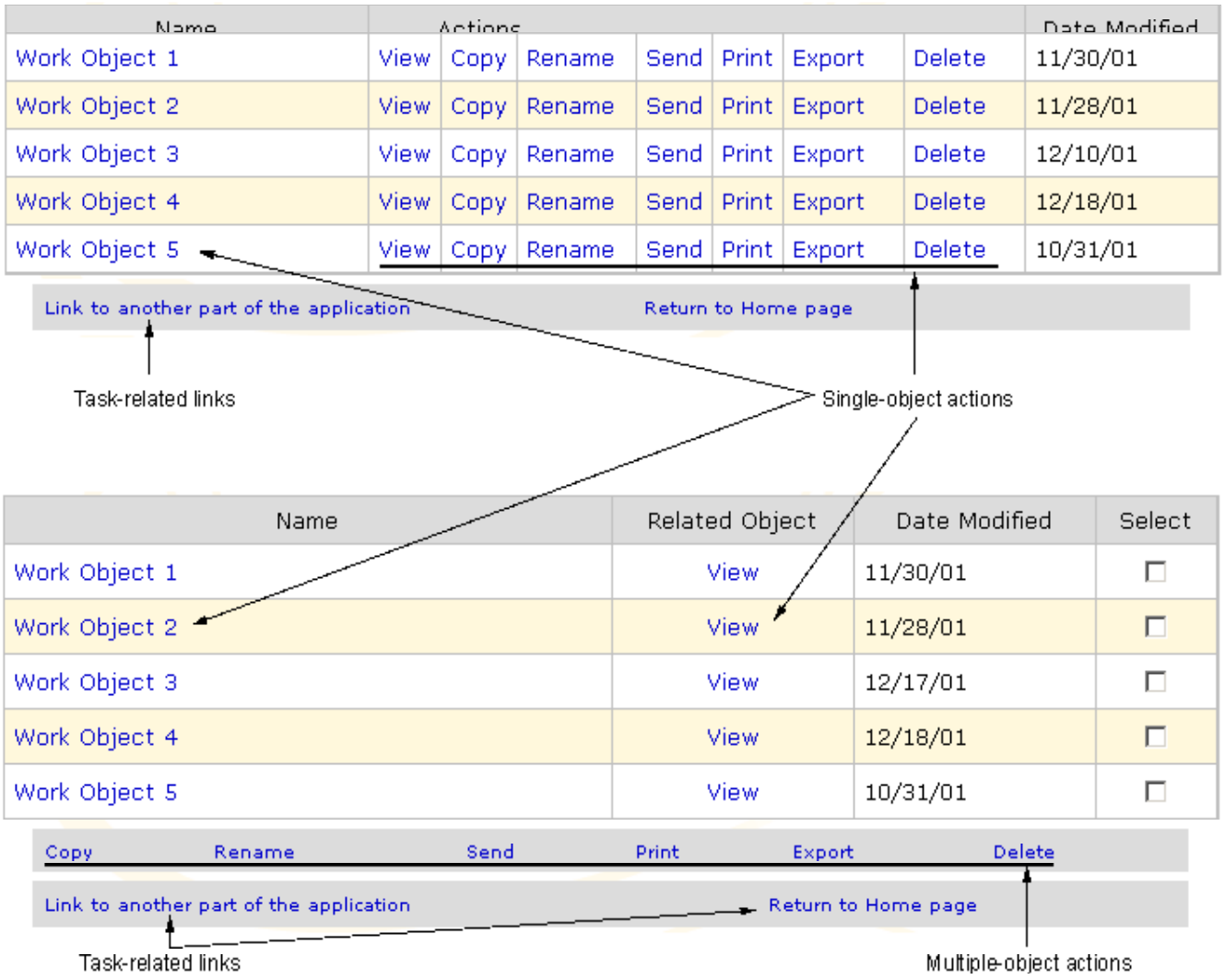

Figure 7. Alternative Table Designs

application prevents the user from interacting with the application, the application should tell the user that they will not be able to continue working with the application until the most recent activity is complete. The purpose of preventing the user from interacting with the application is to prevent anything from happening that will invalidate the work being done on the server. This strategy should only be used if the delay is short and the user knows that the delay will be short. This strategy is inappropriate if closing the browser will lose the user's work.

- Give the user a choice of waiting for the process to finish or being notified that the process has finished. This is appropriate when the delay could last a minute or more. It is unlikely that the user will want to stop working. If the user chooses to receive notification that the process has finished, the notification (usually an e-mail) should contain a link into the application at the appropriate page.
- Tell the user how long the process will take to complete. Unfortunately, in many situations, the length of the delay cannot be estimated. Do not display a progress indicator unless the movement of the indicator is proportional to the processing time. Too often, progress indicators frustrate rather than inform users. Because we cannot estimate the duration of delays at the time that a process starts, we are investigating storing processing times for types of activities and giving estimates like "The last time you did this, it took *x* minutes."

Letting the user know when something is done can be a problem. A number of factors determine whether an application may push a new page to the client. Pushing a new page after the processing

stops is appropriate when the user has indicated, explicitly or implicitly, that he/she is waiting for the results of the processing. Pushing a new page is not appropriate if the user has navigated to another part of the application. If the user is able to navigate elsewhere after starting processing, there should be an easy way to find the results of the process.

BUTTONS OR LINKS

Web applications contain both buttons and links. We have rules for deciding when to use a link or a button. Links are for navigation to other parts of the application when navigation does not cause any other action to be performed. Buttons are for performing actions when the application can perform the action without further input from the user. In most cases, these actions cannot be stopped once they are started. Performing an action may take the user to another part of the application but only when it makes sense within the context of performing the action started when the user clicked on the button. Buttons with ellipses following the text in the button label initiate actions that require more information from the user. Clicking on the button does not commit the user to carrying out the action. So far, we've only used these buttons when a system window (for example, **Print**, and **Export**) is part of the application. Actions may also appear in a drop-down list box. Selecting the item in the list box does not initiate the action. These list boxes are paired with a **Go** button. Clicking on the **Go** button initiates the action.

LINK LABELS

In Windows GUI applications, command buttons that open new windows are supposed to be labeled with the name of the window they will open. Button size constraints and window-naming difficulties have often made this guideline one of the more restrictive ones. Fortunately, Web application design has given us the opportunity to compare the window-name approach to button labeling to an alternative, task-specific approach to labeling. Following the window-name approach to labeling, navigational links in a Web application are labeled using the name of the page that is the target of the link. In contrast, task-specific labels describe the action the user probably wants to perform at the point that the link is available. Testing suggests that task-specific labeling works better than page-name labeling of links.

NAVIGATING WITH LINKS AND BROWSER CONTROLS

It should come as no surprise that people in our usability tests use the **Back**, **Forward**, and **Refresh** buttons and the History window when interacting with Web applications. We believe that users should not need to use the browser's navigation controls when interacting with a well-designed Web application. There should be navigational links on every page that take the user to the places in the application that they want to go. This does not mean that every page links to every other page. It means that the links support the tasks identified during requirements gathering in a manner described by the task analysis. Our experience also shows that people will use the browser controls even if there are links in the page that will take them to the same place as the browser controls. Even in those situations when we have tried to prevent people from using the browser controls, they have gone through the effort of finding them and using them instead of using the application's controls. The following guidelines are recommendations to make navigation in the application easier:

- Always provide links to the pages users will need to visit. This may seem obvious but some application designers believe that the browser controls are part of their application. Do not force the user to rely on the browser controls; for example, there should always be a link that is equivalent to the browser's **Back** button.
- Ensure that the application work wells with the browser since there is little chance that users will only use the links inside the application. If you find that people are using the browser controls instead of the application's controls, it usually means that there is a link missing from the page or that the link is not noticeable.
- Give every page a meaningful name so that people can use the History feature effectively.

CONCLUSION

Unlike with Windows applications, which can rely on the Windows style guides and user interface guidelines, there are currently no formal standards for Web site design. Although there are many sites that offer guidelines, there is as much variation as there is standardization. Many of the design problems we are confronted with are not addressed by any guidelines because of the complexity of the areas in which SAS users work. We continue to investigate new designs while standardizing the appearance and interaction behavior of Web applications.

CONTACT INFORMATION
You may contact the author at:

Todd Barlow
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Work Phone: (919) 531-3031
Fax: (919) 677-4444
Email: todd.barlow@sas.com