SPIDER CONTROLS:

A New Paradigm for the Formatting and Placement of Menu, Toolbar, List-Combo Box & Dialog Box Controls

by

Stanley W. Driskell, PhD

Appendix by

Clif Flynt

ABSTRACT

Spider Controls holds the promise of transforming the ways in which people interact with computers by offering a new paradigm that is both more user-friendly and more productive than the 30-year old computer-human interface that currently dominates the world's computers. Spider Controls is built on a universal graphic suitable for display of the menu, toolbar, list/combo box, dialog box, and palette controls. Although Spider Controls incorporates more than 20 innovations, four are specifically designed to enhance user productivity relative to users of traditional controls. First, the remote-selection feature enables the user to select any icon from a toolbar of over 40 icons after a total traverse of 0.5 inch. Second, when remote-selection is employed in conjunction with the multiple-selection feature the user can often make two selections with a single 0.5-inch traverse. Third, the central area of the Spider graphic contains buttons that permit performing several control management functions that include three ways to exit Spider plus easy redisplay of any ancestor or child control. Finally, the traditional error prone scrolling of list and combo boxes is replaced by paging that enables users to display any of 70 items in a combo or list box list via a single traverse of 0.4 inches followed by a single click. Special purpose buttons that appear in the central area of the graphic achieves this efficiency.

Spider has two additional features that offer great benefit to the user. First, at activation, Spider pops-up the desired control at a predictable location near to but not obscuring the user's area of screen interest and simultaneously jumps the cursor to the control's center. Spider subsequently jumps the cursor to the center of each control as it is displayed, which allows the option selection after the aforementioned 0.5-inch traverse irrespective of the type of control displayed. If the user does not choose to override the feature, when Spider closes the cursor is jumped to the location it occupied when Spider was activated. These features allow the user to avoid the long traverses to and from the screen top so common with the traditional interface.

For software developers interested in adapting Spider to their own applications, Spider Controls offers the “Spider Toolkit”. The toolkit is built on an initial set of Tcl widgets that help define all options available to the controls of the application being developed. Additional widgets enable the user to generate the Spider graphic appropriate to each control to be created. Finally, populating the Spider shell with an option subset is performed by drag-and-drop from the sets of previously defined options. Should the application require that some controls contain embedded controls, special Tcl widgets assist with their generation and positioning. During integration of a completed set of Spider formatted controls into an application, separate routines enable the designer to calculate the display position, to handle ancestor-child redisplay, to manage list-combo box paging, and to pass requests for functionality to the application.

Use of Tcl to build Spider Controls and the Spider Toolkit can make development of Spider Controls and the Spider Toolkit less difficult than likely with alternate approaches.

Overview: The Traditional Human–Computer Interface

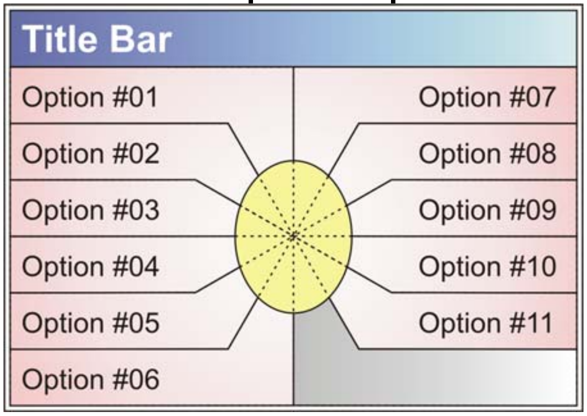
Software developers and users often measure progress by the addition of new functionality. While not belittling the importance of these advances, additional functionality to mature software frequently adds little of value to any but the most specialized users of this software. This paper explores the less frequently visited but equally important area of software usability through the introduction of Spider Controls[1]. Spider Controls is a more productive, ergonomically improved replacement for the currently dominant computer-human interface based on permanent display of the horizontal main toolbar and main menu. Research undertaken during the late 1970's at the Xerox Palo Alto Research Center led to introduction on the short-lived Star computer of what was to become the traditional computer-human interface. With little change, Apple Computer "borrowed" the Star interface for use with the Macintosh. In a second generation of "borrowing", Microsoft introduced most of the Star concepts into Windows. This new interface did greatly reduce memory load relative to command-based systems and led to the large-scale acceptance of the modern PC. However, this interface also saddled future generations of computer users with frequent long traverses, numerous traverse errors, and inconvenient display locations.

During the quarter century since its introduction, development of the computer-human interface has progressed in amazingly *insignificant* ways. By contrast, over this same period, central processor speeds of PCs have increased from a few kHz to over four Ghz. The capacity for mass data storage has undergone a similar transformation. 1-2MB of storage once cost hundreds of dollars, while 250 GB disks can now be found for less than $100. The corresponding development of the computer-human interface is long overdue. While not achieving the levels of Moore's Law, this paper introduces the Spider Controls (commonly referred to as "Spider") technology to provide very substantial usability gains in the GUI

The Spider Controls Paradigm

The basic graphic underlying Spider Controls comprises a title bar (blue), regions (pink), and a "Navigation Center" (yellow). In this figure, the navigation center is shown without content. The Spider display can be summoned with the click of a special mouse button, the stroke of a special key, or a gesture on the touchpad. When the Spider display pops up it appears at a location near to but not obscuring the screen area of current interest to the user. The cursor is jumped by the Spider system to the center of the Navigation Center. When a single option is displayed per region—as with menus and list/combo boxes—the vertical presentation of options permits the leftmost symbols of two or three options to simultaneously encode on the

Figure 1
Basic Spider Graphic



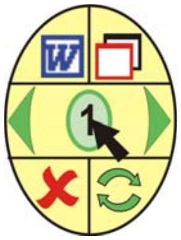
---

[1] The following patents protect intellectual property rights of the Spider Controls system: 5,596,699, 5,880,723, 6, 239,803, 6,801,230, and 6,883,143.

fovea per eye fixation. Parallel cognitive processing of these encoded symbols subsequently permits more rapid option identification than possible with horizontally arrayed controls which only allow a single option to be encoded per eye fixation. Once a desired option is identified, the Spider format permits its selection via a short radial traverse. The fewer, shorter traverses executed with Spider alleviate both arm and shoulder pain and lowers eye strain. Spider Controls thus not only reduces time spent per control activation but also promotes sustained productivity later in the workday by reducing fatigue.

Extended Fitts' Law (www.chimetric.com/technical/ExtendedFittsLaw.pdf) details a method for objective quantification of the physical effort expended to acquire arbitrary polygons. This Law predicts that minimum physical effort is expended to select a Spider region by clicking in its converging area. Additionally, it predicts that equal physical effort is expended in selecting a Spider region irrespective of where the click occurs within the converging area. Two formal experiments were conducted validating these predictions. With the user indifferent to the location where the click occurs within a region's converging area, the inner portion of these converging areas can be allocated for alternate use without increasing the physical effort expended to select a region. Spider Controls utilizes this area to provide a "Navigation Center" with several buttons offering control management capabilities unavailable to traditional controls.

Figure 2
Navigation
Center



The recommended standard Spider Navigation Center has the seven-button configuration shown by Figure 2. Upon activation, the cursor is jumped to the center of the green "multiple-selection" button. When initially displayed, a control is in single-selection mode and remains in that mode unless the button is clicked. When a selection is made in single selection-mode, either the child control linked to the selected option is displayed or the function linked to that option is performed. When the link is to a function, it is immediately performed, the Spider display closes, and the cursor automatically jumps back to the position occupied when Spider was activated. A click on the multiple-selection button converts the control to multiple-selection mode, which permits an arbitrary number of selections. When multiple-selection mode is entered, the multiple-selection button and title bar become filled in red. Selections are declared complete by clicking the "Done" (upper left icon) or the "Cancel" (lower left icon) buttons or by double-clicking the final selection. When the control displayed has a parent and/or child control, the triangular green icons become active and emulate "Back" and "Forward" browser capabilities for redisplay of the parent and child of the current display. The green, circular arrow icon toggles between display of the main menu and main toolbar at Spider activation. The upper-right icon is reserved for a function that is introduced below.

Figure 3
Spider Toolbar

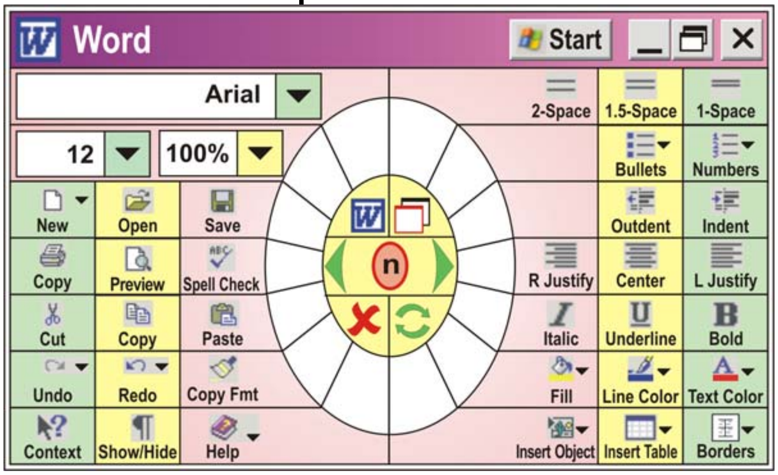


Figure 3 shows a typical Spider Toolbar for Microsoft Word. Although an icon may always be selected by clicking directly on it, the user can also select the primary (green) icon via a single click in the white area of the desired icon’s region. A double click in the white area selects the secondary (yellow) icon. All other icons in a region are selected by clicking directly on them. When a toolbar is in single-selection mode, remote-selection from toolbar having three icons in each of fourteen regions permits the user to select any of 42 icons possible after a <u>total</u> traverse of approximately 0.5 inches followed by a single or double click. With the toolbar in multiple-selection mode, the user can frequently perform “n” selections with fewer than “n” traverses through use of remote-selection. Spider toolbars, palettes and, frequently, dialog boxes can also offer a “Remote-Selection” capability to further shorten traverses.

Rather than the error-prone scrolling used by the traditional list and combo boxes that is so disliked by many computer users, Spider Controls employs paging. Paging entails dividing a given item list into pages that contain the same number of items with the possible exception of the final page. The number of pages and initial number of items per page is determined by an algorithm that divides the number of items in the list by some maximum number of items permitted per page. Research shows that users consider the desirable maximum number of regions of a display to be somewhere between 12 and 16. Consider the following examples. With 14 teams per display, the 21 teams of the Australian National Soccer League will have one page of 14 teams and one page of seven teams with seven inactive regions. At 12 states per page the 50 American states will occupy four full pages plus a fifth page displaying Wisconsin, Wyoming and ten inactive regions. A list of 162 fonts of 14 fonts per page requires 11 filled pages with a final page containing eight fonts and six inactive regions. Extended Fitts’ Law shows that the physical effort of clicking the converging portion of a Spider region is inversely related to the subtended angle. Thus, to minimize physical effort expended per selection, Spider Controls employs an algorithm that maximizes the converging angle by allocating empty regions of the last page among the other pages. After optimization, the Australian soccer teams will be displayed with a 12 region graphic; i.e., 12 teams appearing on page #1 with 9 teams appearing on the page #2. The 50 American states would fully occupy five pages of ten states per page with no unused regions on the last page. No adjustment is possible with the font list since there are fewer inactive regions on the last pages than total pages.

Figure 4
Spider List Box

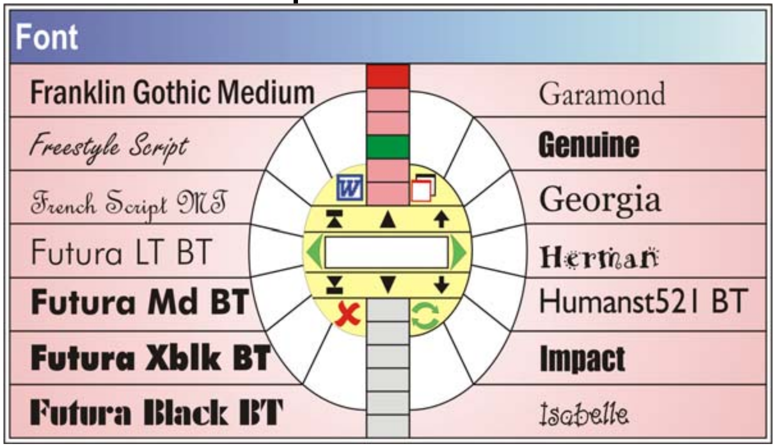
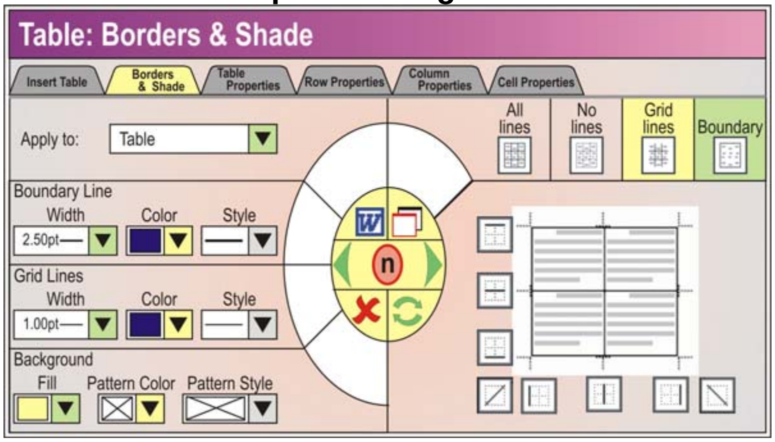


Figure 4 depicts the font list box displayed by single clicking the remote-selection area of the upper-left region of Figure 3. Two features have been added to the Navigation Center to expedite display of the page containing a desired font. Ordered, rectangular “page-blocks” extend above and below the central oval of the Navigation Center. Each page block identifies a page of the font list. If the page containing a desired item is known, that page can be displayed by clicking on its page block. Alternately, if the name of font is known keying its left most symbols into the search box appearing between the forward-back buttons displays the first page that contains the symbol string. To minimize the number of necessary traverses, Spider Controls dynamically generates up to six page-management buttons within the Navigation Center. For lists of less than 42 items (2 or 3 pages of 14 regions) the triangular page-up/page-down buttons are provided. With lists of 43-70 items (4 or 5 pages of 14 regions) barred triangle list-top/list-bottom buttons are added. These four buttons permit display of <u>any</u> item from lists of up to 70 items with a single 0.3-inch traverse and a single click.

The final two page-management buttons provided by Spider Controls are the binary-up/binary-down buttons. These are displayed as directed arrows, and appear when lists exceed 70 items. At initial display of the aforementioned font list box, the recent selections page appears as the red filled top page block. Since wrapping of page blocks is supported, a click on the up binary button displays the middle page of all pages. A second click displays the middle page among the pages preceding the currently displayed page. Figure 4 depicts a Spider List Box display with all six page-management buttons present. Page blocks are color coded to denote a page block as being inside (pink) or outside (gray) the binary search range. When only the binary buttons are clicked, the search range is determined as with any binary search. However, since page-management buttons may be clicked in any order, a click on a non-binary button generates a binary search range comprising pages between the prior and the new current page. Thus, in Figure 4, a click on the list-bottom button results in page block #12 having green fill, page blocks #2, #3, & #4 having gray fill with the rest having pink fill.

Figure 5 illustrates the Spider Dialog Box. Its most consequential feature is its utilization of the basic Spider graphic to permit management of dialog boxes the in same format as the other Spider controls. This makes it unnecessary to remove displayed submenus before displaying a dialog box. Tabs are also supported. In addition, regions of the Spider graphic permit natural grouping of related embedded controls. By declaring the height of the rectangular portion of the regions greater than that portion in other controls, space becomes available for labels that identify groups of embedded controls. If a unique configuration such as the lower right-area of Figure 5 is required, regions can be concatenated and portions of the remote-selection area suppressed.

Figure 5
Spider Dialog Box



Apply to: Table

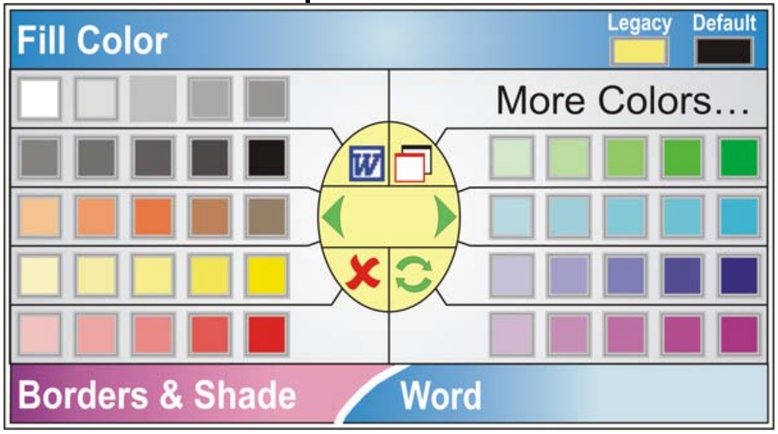
Boundary Line: Width 2.50pt, Color, Style

Grid Lines: Width 1.00pt, Color, Style

Commonly, a given value for an embedded control will be employed during several successive uses of a dialog box. Spider Controls provides users with “legacy” values that override the “default” values of embedded controls pre-selected by system designers, which reset to that default after each use. Legacy values allow the user to retain a desired value until explicitly changed.

While a Spider Palette is a variant of the Spider Toolbar, a palette generally assigns a parameter rather than executing a function, as is typical of the toolbar. Figure 6 introduces two features of Spider Controls that are not confined to the palette. Legacy values were

Figure 6
Spider Palette

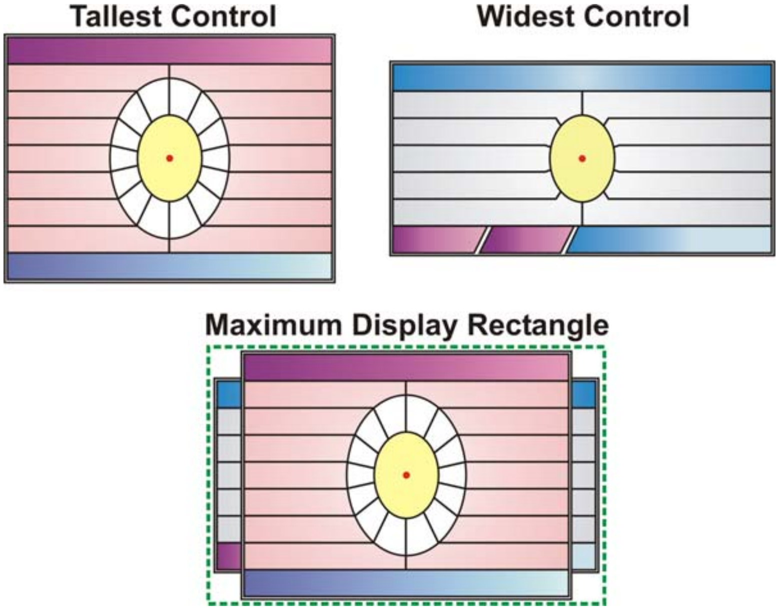


introduced during the discussion of the Spider List Box above (see Figure 5) without indicating how these values are set. Note the title bar of the "Fill Color" palette control of Figure 6 contains both "Legacy" and "Default" edit boxes. Under Spider Controls, a control containing an embedded control displays the value in the legacy box of the embedded control. Thus, if clicking the "Background Fill" option of Figure 5 generates the Figure 6 display, the color displayed in the background fill box of Figure 5 is the color in the legacy box of Figure 6. Although the legacy box contains the default value when the application is first activated this value can be changed in three ways. First, a drag-and-drop of any region value to the legacy box alters the legacy value. Also, when appropriate, a desired legacy value can be directly keyed into the legacy box. Finally, users can drag-and drop from the default box to the legacy box, which resets the legacy value to the default value.

The second feature illustrated by Figure 6 is the "Ancestor Control Bar" abutting the bottom of the currently displayed control. When a new current control is displayed, the center of the new control's Navigation Center is positioned on the coordinates of the center of the parent control's Navigation Center and the parent's name is displayed as a tab at the left of the ancestor control bar. This tab is color coded with the selection state of the ancestor. Shifting ancestors rightward maintains the generational order of ancestors along the ancestor control bar. In Figure 6, "Borders and Shade" is the name of the parent of the current control "Fill Color" while "Word" is the grandparent of "Fill Color." This ancestor management strategy reduces error by reducing superfluous information from peripheral vision that would occur if ancestor control identification comprised more than the control's name and selection state. This configuration also minimizes eye movement while achieving parsimonious consumption of screen real estate.

Research shows that users prefer to have controls displayed at a predictable location close to but not obscuring the screen area of current interest. To meet these positioning recommendations, Spider Controls employs the concepts of Maximum Display Rectangle (MDR) centered on a fixed-point. An algorithm determines the dimensions of the rectangle needed to contain the tallest and another to contain the widest controls that can be displayed. Although it is possible, the widest control usually will not be the tallest control. Dimensions of the MDR are dimensions of the bounding rectangle when the centers of the Navigation Center of the tallest control and of the widest control are superimposed. The MDR of Figure 7 is illustrated by the green rectangle. To determine the optimum MDR position,

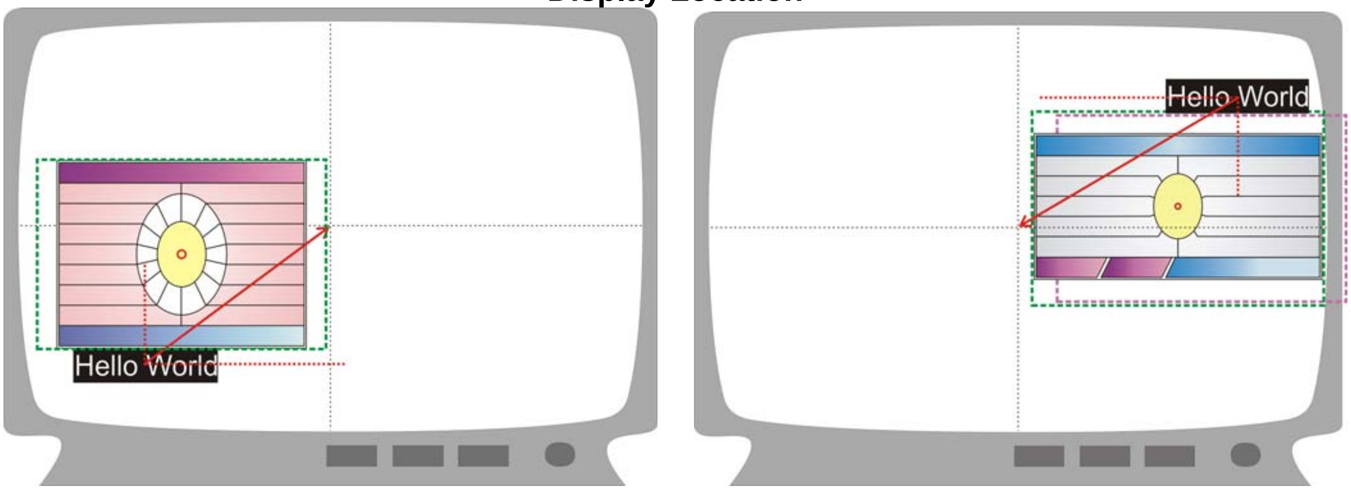
Figure 7
Determining the Maximum Display Rectangle



Spider defines the Target (T) to be the screen area of current interest. When an area of the screen is highlighted, dimensions and coordinates of the center of T are explicitly available. When highlighting is unavailable, T is implicitly defined as a rectangle of predefined dimensions centered on the cursor hotspot. To appraise different shapes of T, define the Aspect Ratio (AR) of T as: $AR_T=(T_{height}/T_{width})$. With applications such as word processors that commonly have $AR_T <<1$, the top or bottom of MDR is positioned to abut the bottom or top of T respectively.

If T is in the lower half of the active window, as illustrated by the left panel of Figure 8, the optimal vertical coordinate of MDR, Y_{MDR}, is: $Y_{MDR}=Y_T + 0.5(T_{heighth}+MDR_{height})$. The corresponding X-coordinate, $X_{MDR,}$ is a predefined percentage, P, offset of one-half of T_{width}; namely: $X_{MDR}=X_T+k0.5PT_{width}$, where k=1 when X_T is left of the vertical center of the application window with k=–1 when X_T is right of the vertical center. The location (X_{MDR},Y_{MDR}) is the “Fixed Point”, since the center of each control displayed is set these coordinates.

Figure 8
Display Location



The right panel of Figure 8 depicts a T with its center above the center of the application window. Y_{MDR} is now calculated as $Y_{MDR}=Y_T - 0.5(T_{heigtht}+MDR_{height})$. If the optimum positioning of MDR for this T is employed, the MDR extends beyond the application window’s boundary. This is shown by the maroon rectangle. If this condition is found during Spider

Controls tests, the coordinates of the fixed point are translated to place the offending MDR edge on the boundary of the application window.

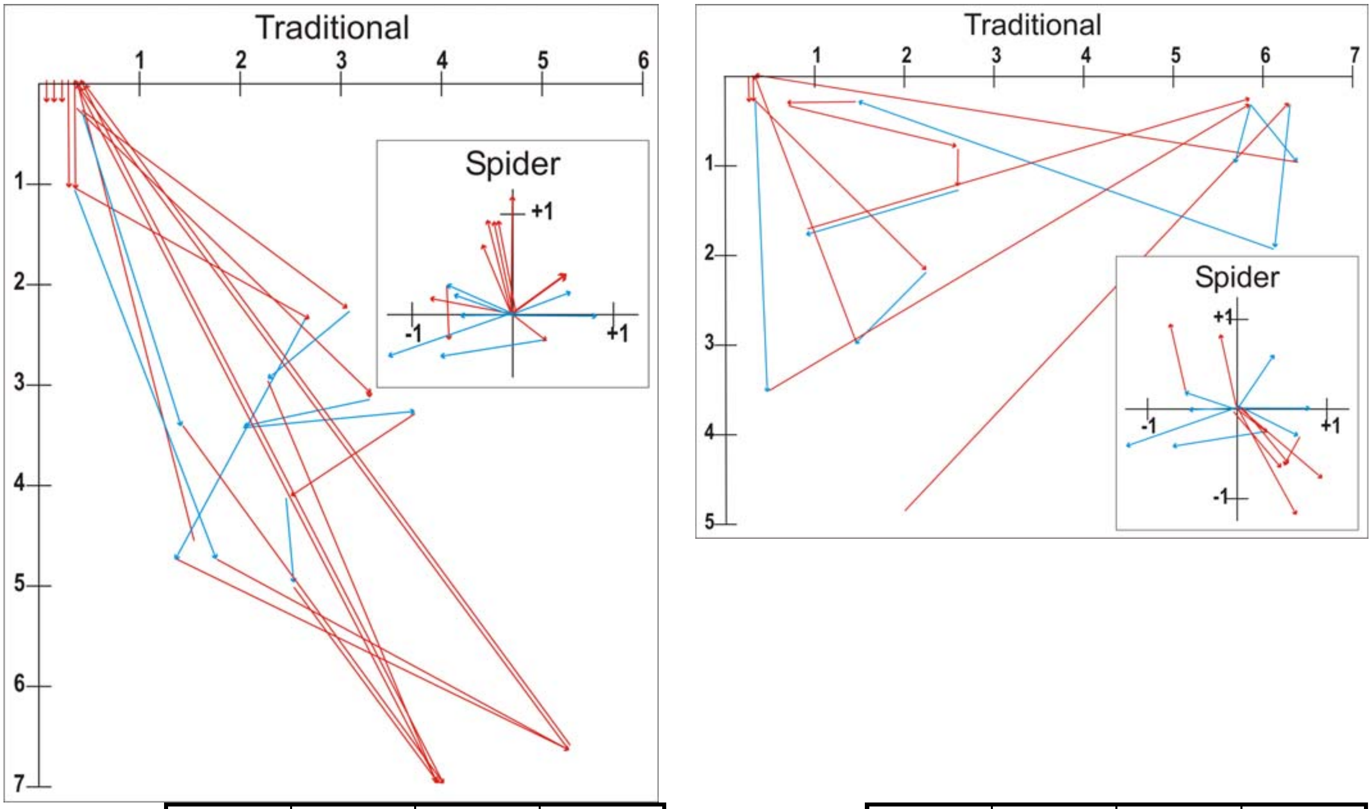
When this logic determines the location of each display, Spider Controls meets the three key recommendations from the professional literature for positioning control displays. First, the display is always predictably located toward the center of the application window. In addition, because it abuts T, a maximum control cannot be located closer to T without violating the admonition of not obscuring T. When a control is displayed that is not maximally dimensioned, the display is still close to T. For example, the edge of a ten-region display is only 0.25 inches from the edge of T. Finally, the fixed point achieves the additional benefit of minimizing eye movement.

Efficiency of Spider Controls

Out of all of Spider's features, the multiple and remote selection capabilities in conjunction with the initial and terminating cursor jumps provide the largest increase in productivity relative to traditional controls. This productivity gain is visually suggested by Figure 9, which

Figure 9

Comparisons of a Seven Step Word Editing Task



Menu Based Performance

Toolbar Based Performance

	Service Traverse	Function Traverse	Distance Estimate	Time Estimate
Spider	9	7	36.2	9.0
Traditional	19	7	148.9	28.6
Rel Efficiency	0.47	1.0	0.24	0.32

	Service Traverse	Function Traverse	Distance Estimate	Time Estimate
Spider	7	6	20.4	6.7
Traditional	11	7	67.2	16.8
Rel Efficiency	0.64	1.00	0.30	0.40

depicts the number and length of traverses executed to perform a seven word editing task comprising the following steps: (1) set font to bold, (2) change font, (3) change font color, (4) color fill the background, (5) animate the background, (6) change the background fill color, and (7) change the background animation. For this demonstration, these seven tasks were performed twice with each interface design: once with maximum menu use; once with maximum toolbar use. Direct screen capture provided a visual representation of the number and length of traverses executed when using traditional controls. Storyboards drawn to the scale proposed for a Spider Control implementation provide a visual representation of the number and length of the Spider based traverses necessary to complete the same seven tasks.

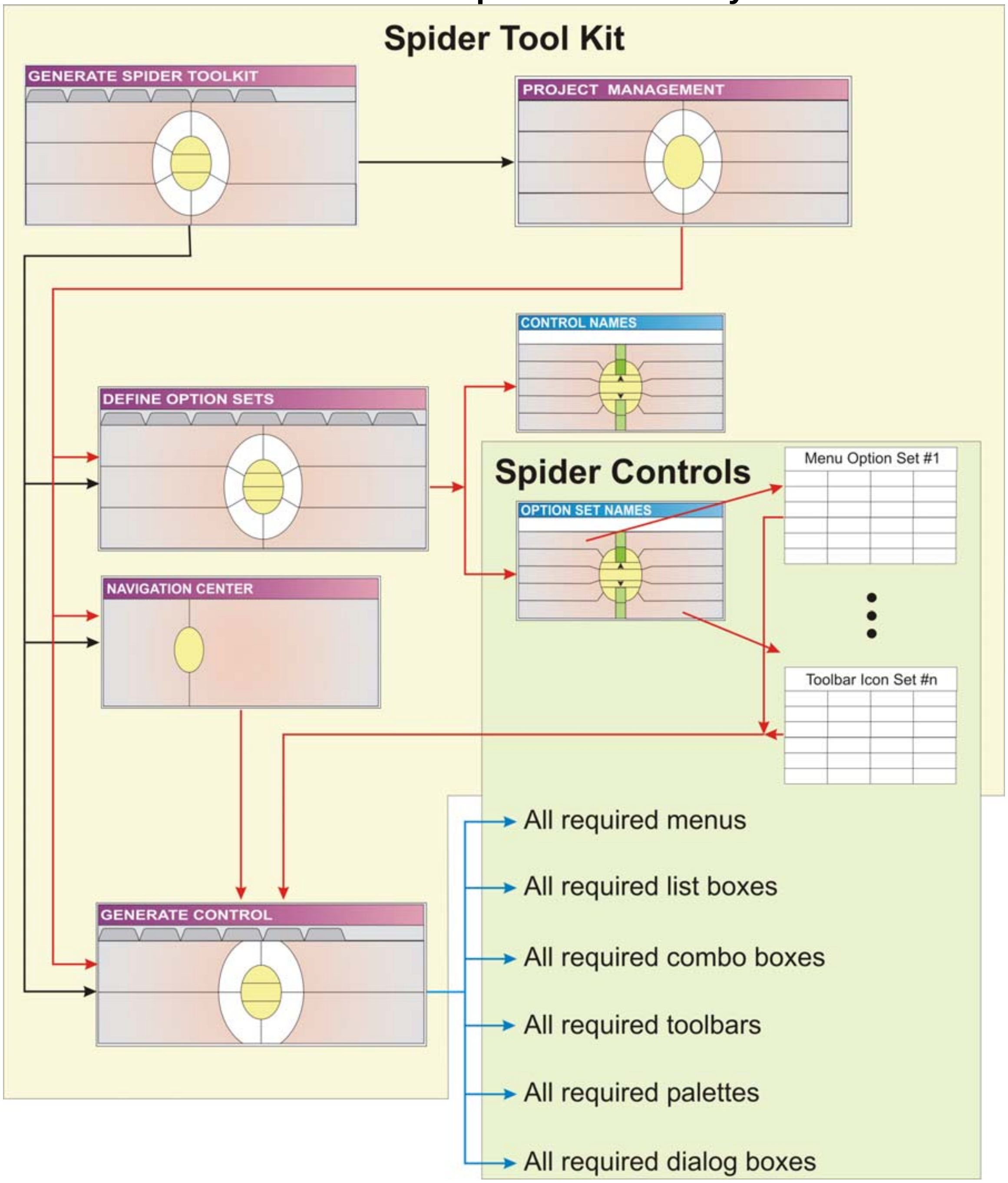
The red traverses in Figure 9 identify support traverses executed to display controls containing desired functions. Blue traverses identify those that initiate a desired function. The need for fewer support traverses generally indicates a more desirable interface. Because the Navigation Center permits the redisplay of any controls irrespective of type; Spider Controls will always have a substantially lower ratio of support controls to total traverses than will the traditional system. In multiple-selection mode, the upper-right icon of the Standard Navigation Center permits the speedy redisplay of the most recent ancestor in multiple-selection mode while automatically jumping the cursor to the multiple-selection button of control as it is displayed. This feature commonly permits the user to perform the next selection without first needing to recreate the path to an ancestor control. Since the selection of a dialog box from a traditional menu closes that menu and any ancestor menus, users of the traditional interface frequently must reproduce the support traverses necessary to redisplay a menu containing a previously accessed set of functions.

The tables accompanying Figure 9 show that the fewer, shorter traverses required by Spider Controls enable the user to accomplish a given task-set in less time than with traditional controls. These time estimates derive from a simulation program based on Extended Fitt's Law that tracks the position of the user's arm rather than cursor position. Traverse distances shown are thus the aggregate distances moved by the elbow, forearm, wrist, and fingers. The values shown are scaled by the multiplicative factor between mouse travel and cursor travel and thus are not the actual distance moved by the cursor. While the distances in Figure 9 do not directly relate to cursor movement, the ratio of physical effort expended when using the Spider and traditional systems is directly reflected in the amount of arm movement necessary and therefore directly related to the time required to perform a task set. (See: http://www.chimetric.com/spider → Technical Justification→ Simulation Analysis, p. 21-28). Results suggest that when employing maximum menu usage, the Spider based approach performs the seven tasks in about one-third the time required by traditional methods. When toolbar usage is emphasized, Spider performs the seven tasks in about 40% of the time required by traditional usage.

Implementation of Spider Controls

Creating a generalized control system requires that software designers be provided with a convenient method of specifying that the options displayed by the control system be most appropriate for their specific needs. To achieve this, with the exception of informational controls, the control system must assign a defined screen area to each option and link that area either to the generation of a requested control or the activation of a specific function. With the possible exception of the dialog box, the technical demands of providing these capabilities is greater with the Spider format than with the traditional format. To ease the introduction of the Spider interface, the Spider Controls system thus includes the Spider ToolKit to provide software designers the tools necessary to generate any desired control based on a few, consistent parameters. Such easy generation of controls additionally enables application designers and human factors specialists to conveniently interact in order to craft the optimum control system for a new product.

Figure 10
Schematic of the Spider Controls System



The Spider Toolkit performs four primary activities during the generation of the menus, list boxes, combo boxes, toolbars, palettes, and dialog boxes required by a software product. The “Project Management” activity identifies a specific Spider project, sets global parameters for that project, monitors its status, and initiates the activities of the other three tasks. Performing these supervisory activities entails inputs into a single form. The “Specify Navigation Center” activity exists to provide for situations that require navigation controls other than those available with the Standard Spider Navigation Center. Handling this activity entails accessing a single form and selecting from one of six alternate navigation center templates. The designer then inputs icon and text to identify the button plus specifications that identify the functionality assigned to each button. Activity three, “Defining Option Sets” specifies the option sets appropriate for each control. Each option set consolidates related options into convenient groupings to assist the customization of individual controls in order to best meet a specific use of an application. Activity four manages the generation of the required controls. Creation of option sets and control creation each requires one form per control type.

To complete the four activities of the Spider Toolkit thus requires one supervisor form, one Navigation Center form, and two forms for each of the six different control types; a total of 14 forms. Additional forms are required to support the fourteen major forms. These comprise a form for display of option set names and another for display of names of the generated controls. Another specialized form provides templates identifying control types that can be embedded into controls generated by the toolkit. Finally, three additional forms are available to control the positioning of embedded controls and providing the maximum and minimum values found with spin buttons and sliders.

Interestingly, all forms of the Spider Toolkit can be displayed as Spider dialog boxes and thus capable of being generated by the dialog box generating form the Spider Toolkit. The resulting implementation comprises a base capability that generates the forms of the Spider Toolkit that, in turn, generate the controls for a given application software. Schematically, this looks like Figure 10.

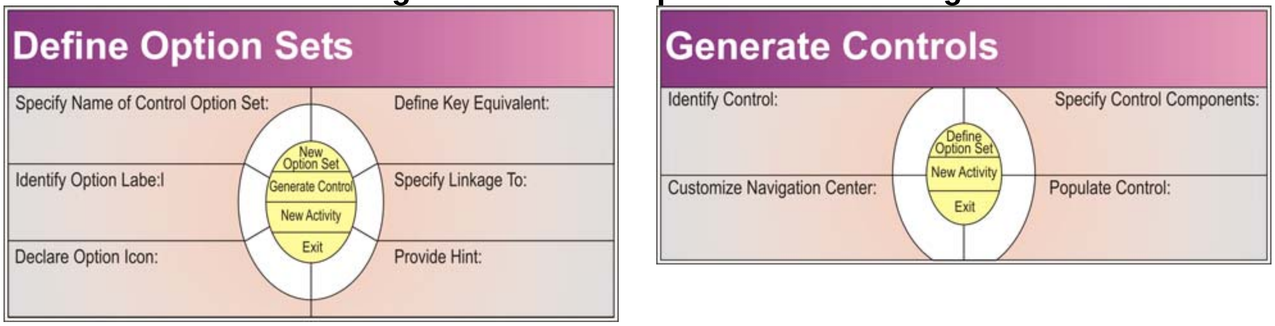
Directed black lines emanating from the “Generate Spider ToolKit” form depicts creation of the major forms that comprise the Spider ToolKit. Red directed lines communicate the most direct way of employing the forms available in the Spider ToolKit to generate the controls. The actual generation of controls for an application is depicted via the directed blue lines. The “Control Names” and “Option Set Names” forms represent a simplification since, as previously noted, one such form will be needed for each control type required by the application. With the “Option Set Names” and “Control Names” forms being unknowable until the needs of a specific application are known, these forms are created by the “Generate Control” form of the ToolKit rather than via the initial “Generate Spider ToolKit” form.

The “Option Set Names” form and specific options subsumed under each name displayed therein are integral to both the Spider ToolKit and Spider Controls. In the toolkit, the “Define Option Sets” form is employed to identify and provide the four parameters requisite to every option. When exercised from within Spider Controls, access to the various option-sets permits customization of each control to achieve maximum efficiency during resolution of the particular problem addressed by the application.

Although traditionally different terminology is employed to identify options offered by the different control types—options for the menu, items for list and combo boxes, icons for the toolbar, etc.—the user ultimately faces different selectable options. Highly specific applications that do not need an ability to customize the menu and toolbars will only require defining one set of options per control. By contrast, users of general-purpose applications – such as word processors and graphic design applications – need the ability to customize the control system with that subset of options needed for the problem at hand. This is achieved by providing the user with several option sets per control; each option set offering capabilities related to an aspect of the problem universe.

The left panel of Figure 11 outlines the approach proposed for defining any number of option sets for any control type. The upper left- and lower-right regions provide support capabilities. Each of the other four regions solicits one of four categories of information required to fully specify each option. Although the granularity is too large to show specifics, the information solicited by these four critical regions is identical irrespective of the control to which the option set applies.

Figure 11
Basic Configuration of Principle Control Defining Forms



The right panel of Figure 11 conveys that only four categories of information are required to fully define any control amenable to Spider format. While not as consistent as defining option sets, substantial consistency exists with all but the “Specify Control Components” region where differences are primarily attributable to complexities of positioning multiple embedded controls embedded in a single region. Figure 5 above exemplifies this. An additional complexity can arise when there is need to adjust regions by subdividing single regions, consolidating contiguous regions, or performing a combination of these. Again, see Figure 5 above. The Spider ToolKit manages this later need for illumination of single and contiguous regions with subsequent display of a pop-up control that solicits information about adjustments desired in the illuminated regions.

Conclusion

Two primary goals are met by the Spider Controls system. First, the extreme attention given to ergonomic principles increases user productivity while lowering physical discomfort of computer usage. Second, the Spider Toolkit brings human factors specialists, who typically possess modest programming skills, more fully into design of the control system. The ease of prototyping different controls during alpha level development can substantially enhance the usability of application software being developed under a tight budget.

Tcl can expedite implementation of Spider Controls. Each of the seven basic Spider formatted controls that can appear in an application is a collection of four rectangles, a variable number of concave pentagonal shapes called regions and an oval subdivided into several buttons. Numerous additional rectangles are incorporated into list and combo box displays. Application of Tcl to Spider Controls involves providing inputs to the Tcl interpreter that result in generation of specified rectangles, the appropriate number of Spider regions, their location, and their relation to each other. Additional code then populates the Spider regions with the differing option types that have been determined during design to most efficiently accomplish the user's purpose. Non-Tcl code of Spider Controls tracks user activity on the interface and when appropriate calls the Tcl interpreter to generate displays and update these displays during their manipulation by the user.

The Spider Toolkit is an application that utilizes Tcl to generate several Spider graphics as permanently displayed forms rather than pop-up controls. The designer uses the "Create Data Sets" group of these forms to specify option sets that will be available to populate the controls. Specifications from the "Create Control" forms enable the designer to specify controls of each type to be created. Tcl then employs this detail to generate the title bar, the cosmetics comprising background and frames, handle variations of the central oval, and determine the number of regions, their height, and content. Once integrated into the application being developed, the output of the Toolkit becomes are the dominate components of the application's control system.

Appendix

A great deal of GUI design can be done with images, focus groups and measurements. However, in the final analysis, you need to put a GUI in front of a user, see what they do, find out what they like and don't like, and then modify the Widget to improve the user experience.

This is where Tcl and Tk come into the picture. Tcl and Tk provide a rapid prototyping environment where user input can be reflected in the GUI relatively quickly and easily.

The Spider widget is essentially a holder widget for other graphic components. The strength of the Spider is how the graphic components are arranged. The radial legs provide a way to organize the components that requires less effort than other GUI layout techniques.

While the grid, pack and place commands are adequate for most layouts, they don't have sufficient versatility for laying out a spider widget. For that reason, rather than making a Spider widget from a frame, the Spider widget is built on top of a Tk Canvas, with commands for creating canvas objects and adding bindings to them as well as placing other standard Tk widgets on the canvas.

The Tk Canvas provides almost all the necessary functionality for creating a complex widget like the Spider widget. The missing functionality, the ability to warp a pointer, can be acquired with a small platform specific extension.

The Spider widget is constructed in a toplevel widget that contains a single canvas. The canvas is divided into legs, and the individual graphic items are placed within the legs.

A leg can be considered as a non-rectangular frame. It's a holder construct, which groups a set of smaller graphic elements. Within the leg, graphic elements can be organized using a grid-like set of row and column coordinates.

A canvas oriented grid package was written in pure Tcl to allow placing graphic objects on the canvas. This simple grid supports placing objects at a specific row and column location, default locations, resizing rows and columns as necessary and anchoring a collection of objects by their north, south, east, or west boundaries.

It does not currently support columnspan and rowspan or a -sticky option for individual graphic components.

The Tcl API for the Spider widget is simple.

Syntax:`spider` *`toplevelName ?-option value?`*

`spider`	Returns the name of a procedure for future interactions with this spider widget
`toplevelName`	The name of the toplevel widget to be created to hold this spider widget.
`-option value`	There are several configuration options that can be set for a spider widget. The non-exhaustive list includes: width

Initial width of the spider. Default value 400 pixels

height
Initial height of the spider. Default value 300 pixels

font
Font to use for labels and the center of ovals. Default is Arial 18 bold.

fadeColor
The center color to fade to white from. Default is #88f

iconFont
A font to use when an image and text are displayed together. Default is Arial 6.

num
The number for the center oval in this spider. By default it is calculated from the number of -prev spider widgets.

next
The spider widget to be displayed if the user clicks on a right-pointing arrow in the center oval.

prev
The spider widget to be displayed if the user clicks on a left-pointing arrow in the center oval.

title
The title for this spider. Defaults to the toplevel name.

upLeft
A list that defines events and bindings when the user clicks the upper left sector of the yellow oval. The format is { {{canvas create command for graphic} {event} {script}} {...} }

upRight
A list that defines events and bindings when the user clicks the upper right sector of the yellow oval. See upLeft for format

downLeft
A list that defines events and bindings when the user clicks the lower left sector of the yellow oval. Default is a red X to close the widget.

downRight
A list that defines events and bindings when the user clicks the lower right sector of the yellow oval. See upLeft for format

centerLeft
A list that defines events and bindings when the user clicks the center left sector of the yellow oval. Default is a left pointing green arrow when there is a previous Spider widget to be displayed.

centerRight
A list that defines events and bindings when the user clicks the center right sector of the yellow oval. Default is a right pointing green arrow when there is a next Spider widget to be displayed.

ovalPctY
Y coordinate for center of yellow oval. Default is .5.

ovalPctX
X coordinate for center of yellow oval. Default is .5.

ovalWd

Width of yellow oval. Default is 60 -width 4 -fill red } {[namespace current]::close}}}

Syntax:$spiderID *createLeg ?-opt val?*

`CreateLeg`	Create a new leg for a spider. Returns a leg identifier.
`-opt value`	An optionname/value pair to define a specific setting for a graphic object to be placed in the Spider. Options include: -bind {bindList} A list of events and scripts to bind to the portion of this leg that has no widgets placed on it. This is normally the primary action for this leg.

Syntax:$spiderID *create type ?-opt value?*

`create`	Create a graphic object on a specific leg of the spider widget.
`type`	The type of graphic object to create. One of the objects that can be created on a Tk Canvas (line, oval, window, image, etc.)

`-opt value`	An optionname/value pair to define a specific setting for a graphic object to be placed in the Spider.
	Options include all of the Tk Canvas options and :
	`-leg $legID`
	The leg to place this object in. By default this is the last leg added to the Spider.
	`-row`
	The row for this object
	`-column`
	The column for this object
	`-bind {<event> script}`
	Define a binding for this object. Events are as defined for the `bind` command. Script is a Tcl script to be evaluated when the event occurs. The script may include % values as defined for the bind command.

Example

```
# Application menu to be invoked by window manager
# when a specified keystroke occurs
# This line in .fvwm2rc will invoke the app on F12
# Key F12         A       A       Exec /usr/local/bin/appSpider
```

```
if {[catch {namespace children ::zvfs}]} {
  set SRCDIR .
}
source $SRCDIR/spider6.tcl
source $SRCDIR/cgrid2.tcl

wm withdraw .

set u [spider .t3 -fadeColor #080 -title {Application Menu} \
    -upLeft {{{create text 10 10 -text "Done" -font {arial 12 bold}}
\
     exit}} ]

set l1 [$u createLeg -bind { {exec soffice &}}]

$u create text -text "soffice" -row 1 -column 1 -leg $l1  \
    -bind {{} {exec soffice &}}

$u create text -text "kwrite" -row 2 -column 1 -leg $l1 \
    -bind {{} {exec kwrite &}}

$u create text -text "emacs" -row 3 -column 1 -leg $l1 \
    -bind {{} {exec emacs &}} \
    -bind {{} {exec emacs [tk_getOpenFile] &}}

set l2 [$u createLeg -bind { {exec xterm &}}]

$u create text -text "xterm" -row 1 -column 1 -leg $l2 \
    -bind {{} {exec xterm &}}

$u create text -text "ftp" -row 2 -column 1 -leg $l2 \
    -bind {{} {exec xterm -e ftp &}}

$u drawSpider
```

Future Development

The Spider widget as currently implemented is useable, but incomplete.

The canvas gridding package needs support for row and column span, and support for a -sticky option for graphic components.

The cursor warping code needs to be merged into a platform neutral framework for Windows, X-11 and MacOS.