

Breaking the Window Hierarchy to Visualize UI Interconnections

David F. Huynh, Robert C. Miller, David R. Karger
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA, USA
{dfhuynh, rcm, karger}@csail.mit.edu

ABSTRACT

Among three fundamental characteristics of contemporary windowing systems—opacity, rectangularity, and hierarchy—the first two have been broken to give way to more expressive UI designs while hierarchy remains unchallenged. We propose a new kind of UI element called links that breaks the hierarchy of graphical user interfaces for the purpose of showing relationships between disparate UI elements. We present a small user study indicating that links are desirable visual elements that are currently under-used in graphical user interfaces. We propose a taxonomy of how links can be used, present a toolkit for representing and displaying links in Java/Swing, and show how links can enhance existing applications' UIs.

INTRODUCTION

Windowed user interfaces currently dominate if not monopolize all existing software applications. Users, UI programmers, and even UI researchers have come to accept that an application's graphical user interface is a tree of large and small, nested, mostly rectangular and opaque windows, overlapping and hierarchically clipped. These characteristics of contemporary windowing systems—rectangularity, opacity, and hierarchy—made sense back in the early days of graphical user interfaces when they bought us performance optimizations that allowed for smooth user interactions. Now, after many years of hardware advances, these optimizations become less and less justified against the demand for expressiveness in UI design.

There is mounting evidence that the traditional windowing paradigm, consisting of opaque, rectangular windows with impenetrable borders, is too confining. UIST has seen continuing research interest in translucency (e.g., [1], [6], and [8] in 2003) as researchers recognize the human ability to parse layered visual content. Non-rectangular UIs also have arrived in the form of pie menus and media players. A recent piece of work [11] explores a circular radar-like interface for showing notifications.

Support for translucency and non-rectangularity has made its way into some windowing systems and is gaining traction. We have witnessed the introduction of translucent windows in

popular media players and instant messaging clients. However, hierarchy remains mostly untouched. It is this third characteristic of the traditional windowing paradigm that we wish to address. This paper explores the concept and usage of links—a new kind of UI element that seemingly crosses from one part of the hierarchical window tree to another part, expressing connections between disparate elements on screen.

We start by redesigning an existing UI to show new possibilities in UI designs when rigid windows are not used. In particular, the redesign illustrates the replacement of selection synchronization by the use of links crossing from one window to another.

Next, we show the results of a user study pilot in which test subjects drew links abundantly when told to design posters but hardly any when designing GUIs for the same purpose. This is evidence for the desirability of links and the difficulty in which they can be programmed in existing UI systems.

We propose a taxonomy of links to understand the nature of links. We then build a prototypical library of link UI elements that can be integrated into Java/Swing applications. We use this library to demonstrate a live version of the aforementioned redesign as well as an enhancement to an existing application through the use of links.

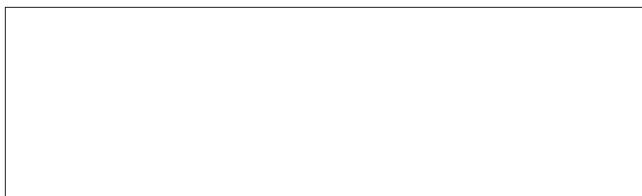
Finally, we discuss related work and future work.

SNAP: A REDESIGN EXAMPLE

In order to illustrate how the aforementioned three fundamental characteristics of the traditional windowing paradigm can be usefully broken, we consider SNAP, a recently developed research user interface [5]. This research system called SNAP provides users with a UI mechanism for coordinating several visualizations that have been rendered from queries to a relational database.

Figure 1 shows a sample UI composed using SNAP. A user loads each of the five windows with data and specifies the formats for rendering them (e.g., plot, map, table, outline). Then, the selections of the windows are tied together using the Snap buttons so that selecting a state in the top left window highlights the corresponding data point in the plot, underlines the state's code in the map, and displays that state's county data in the table and the treemap on the right.

We chose SNAP as an example because of its attempt to let users smash together several visualizations and explore relationships among them. However, the “smashing” is not as



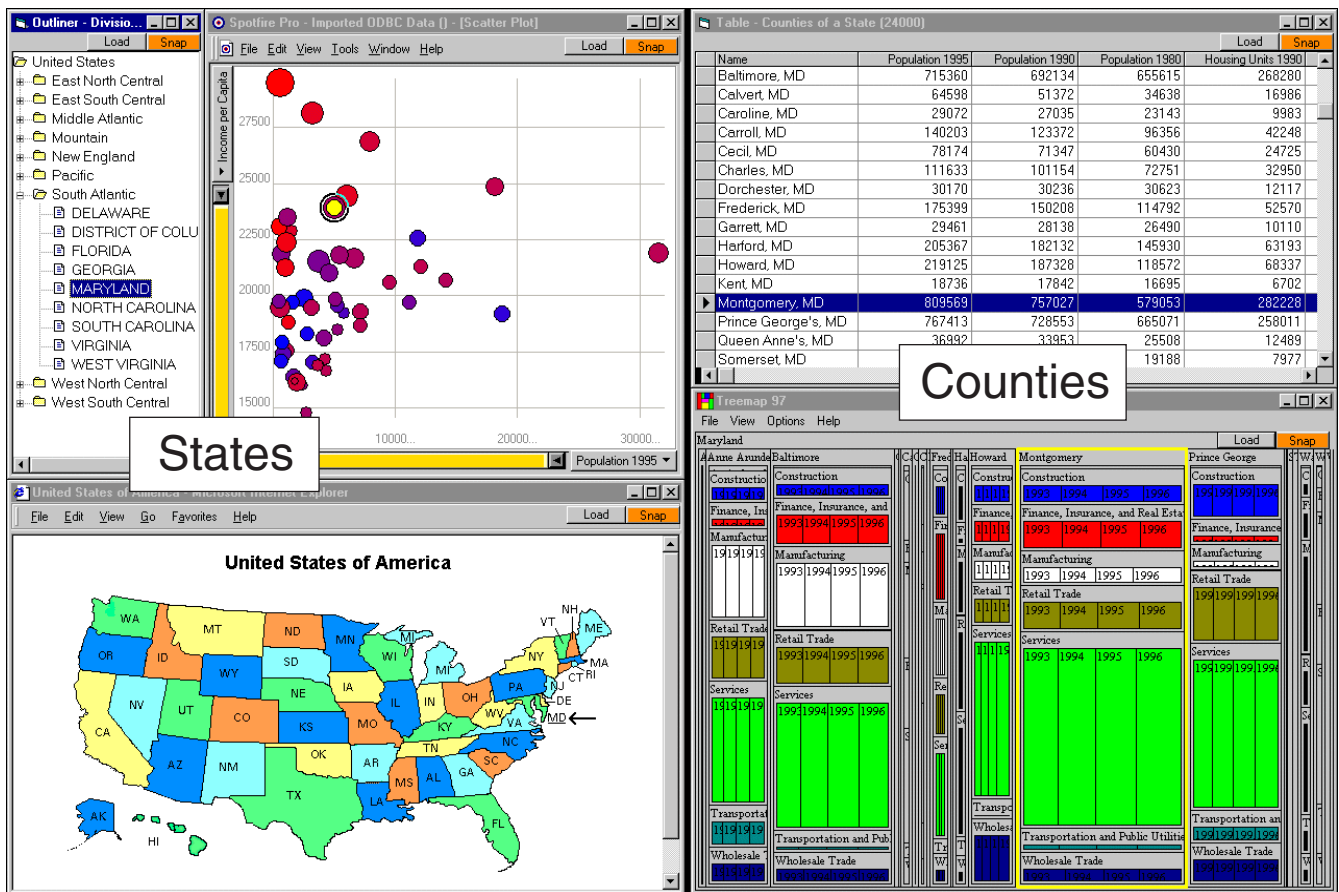


Figure 1. Original SNAP user interface from [North], reprinted with permission from authors

effective as it could have been if windowing toolkits were less confining. Figure 2 shows a rough redesign of the UI in Figure 1. We focus primarily on the information being explored rather than on the UI mechanisms used to specify the visualizations. Note the following differences in Figure 2:

- We use lines to connect the selected state name (e.g., Maryland) to the corresponding state on the map as well as the data point on the plot. We believe that this use of connecting lines, or “links,” shows correspondence more effectively than just highlighting corresponding items in the different visualizations in synchrony. Using most existing UI toolkits, it is next to impossible to draw lines stretching from within one window to another window. Even if it is possible, the resulting links are not first-class widgets and cannot be easily manipulated programmatically.
- Furthermore, since links are used to indicate selection, visual variables formerly used for highlighting are now freed up for other purposes. For example, highlighting can be used to indicate data points on the plot corresponding to other states in the same region as the selected state. This is useful for an overview of how the states in a region are distributed.
- We also use a link to connect the selected state to the county data table. This connection shows not correspondence but elaboration. Nevertheless, since the link attaches to the border of the whole table, users should understand that elaboration is being communicated. There is no longer a need to suffix every county name with “MD” as in the original design.

In addition to using links, we have also made a few other modifications to enhance the UI:

- Most borders are removed. This change allows the plot to be pushed into the map (crossing over its invisible rectangular boundary) without obscuring parts of the map or creating a busy look due to intersecting borders. The two pieces of information are fitted together more snugly. This fitting effectively communicates the relationship between the map and the plot in Figure 2. In contrast, two labels “States” and “Counties” had to be added to the screenshot in Figure 1 to avoid confusion: the plot could have been of county data rather than of state data.
- The column header labels in the table are inclined so that the columns can be pushed closer together. The ability to incline text, difficult in most UI toolkits, adds flexibility to the design of visualizations. Furthermore, note that the bounding boxes of the column header labels intersect one another. As a result, providing a custom header label widget to a standard table widget is not enough—the table widget needs to be able to nudge the header labels closer than a conventional rectilinear layout would allow.

USER STUDY

These interesting areas of the design space remain largely unexplored because of limitations imposed by UI toolkits. In order to get a sense for how the creativity of UI programmers is affected by their perception of what’s easy or hard to implement, we conducted a small study with 10 computer science graduate students in our lab. Each student was asked

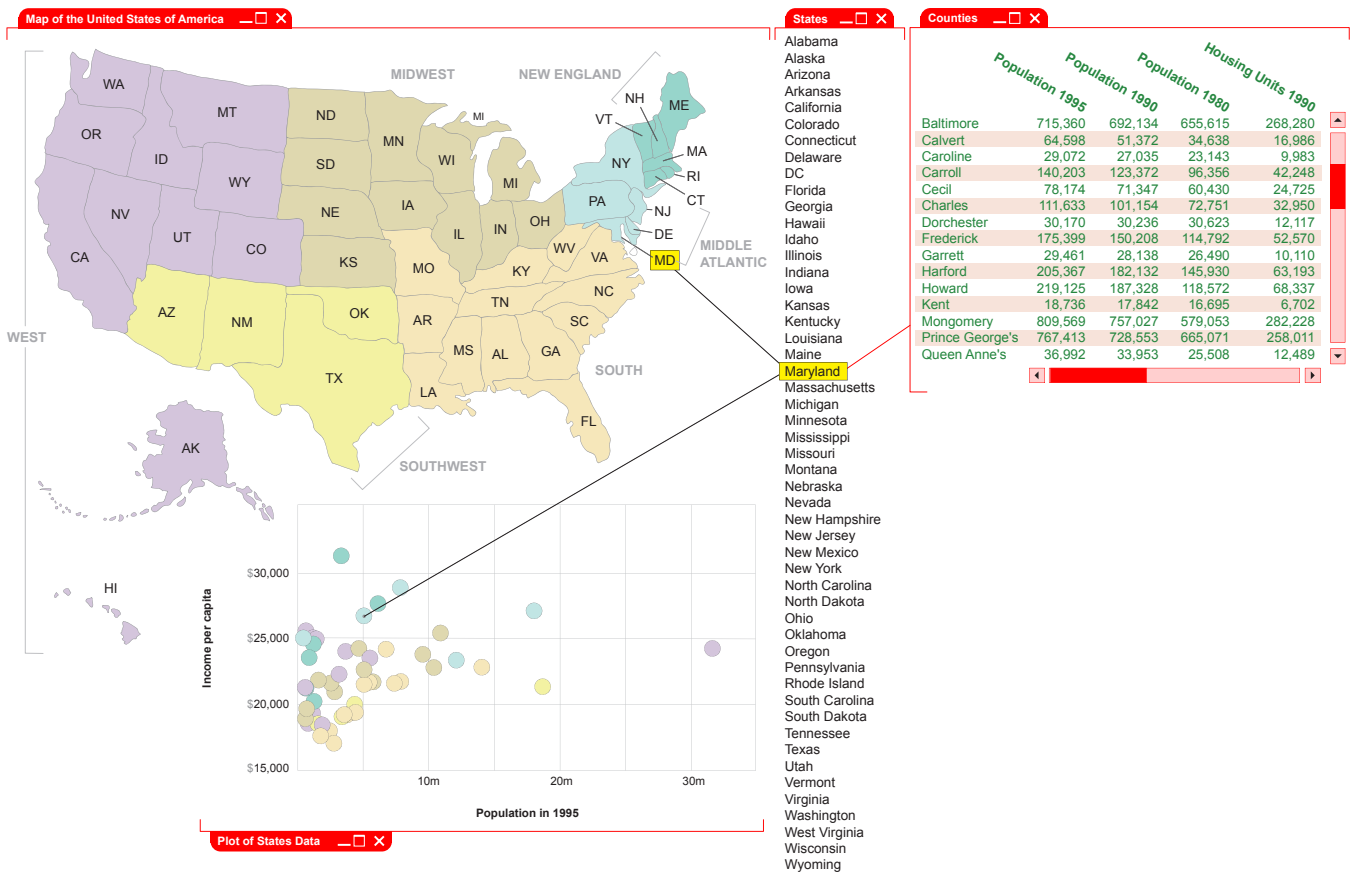


Figure 2. A redesign of the SNAP user interface (income per capita from <http://www.iowaworkforce.org/trends/percapita.html>; population from <http://www.census.gov/population/projections/state/stpipop.txt>)

to design an information visualization, either in poster form or computer form, for the following scenario:

“You have recently gone on a trip to China. During your trip, you visited five cities and stayed a different number of days in each city, and you also made new friends at each city. [Here appeared the list of cities, durations of stay, and pictures of one or two friends made in each city.] As an assignment in your Chinese language class, you are to make a [poster or computer program] to recount your trip to your classmates, which shows (1) your route through China, (2) the friends you met, and (3) the fact that you spent a lot more time in Hangzhou than in any other city. Using paper and pencil, roughly sketch out what your [poster or computer program] would look like. Assume that you have access to a map of China and photos of your friends in any form you’d find convenient, and that the hypothetical class assignment is due in one week.”

We chose this scenario because its solution requires displaying several different visualizations—geographical, chronological, and pictorial—that are closely related. The Hangzhou requirement was particularly intended to motivate creative ways of showing the relationships between visualizations (like Minard’s famous visualization of Napoleon’s Russia campaign [9]). We wanted to see whether there were any systematic differences in how these relationships would be shown in a

paper display (a poster) compared to an interactive display (a computer program).

For 4 of the 10 designers (randomly chosen), the requested visualization was a poster; the remaining 6 designers were asked to design a computer program. All 10 designers had designed and implemented at least one substantial user interface, and had experience with a wide range of UI tools (chiefly Java Swing, HTML, SWT, and Visual Basic). Designers in the computer-program condition were told that they could imagine designing for any UI programming environment that they felt comfortable with. Designers were given as much time as they wanted to produce a design; in the end, all designers took less than 30 minutes. Three designers (2 in the poster group and 1 in the computer group) iterated their designs, throwing away a partial design and presenting a second sketch instead. The other designers created only one design.

The final designs were varied, but some general conclusions can be drawn. First, lines connecting different visualizations were commonly found in poster designs (3 out of 4) but rarely in computer designs (only 1 out of 6). Lines were used to connect cities on the map to friends’ photos (in 2 designs), and cities to points on a time line (2 designs). Several solutions used two kinds of lines, one kind to show the route around the map (a chronology within the geographic visualization), and another kind to make links between visualizations.

Second, most of the poster designs (the same 3 out of 4) were unified in the sense demonstrated by Figure 2: different visu-

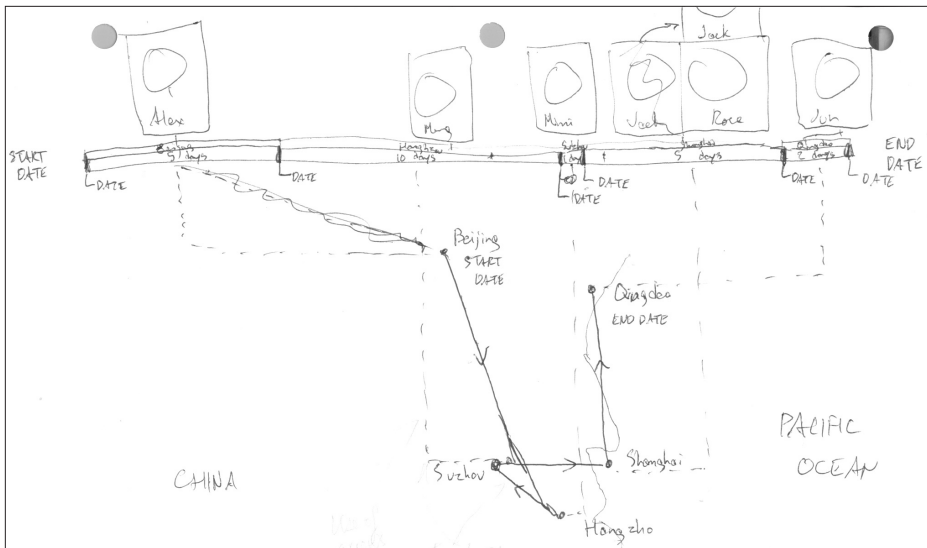


Figure 3. One of the poster designs from the user study



Figure 4. One of the computer program designs from the user study

visualizations freely overlapped without respecting rectangular boundaries. In most of the computer designs (5 out of 6), by contrast, the visualizations were walled off from each other by strongly drawn rectangular boundaries. In fact, half the computer designs (3 out of 6) put different visualizations on different pages, giving up any hope of unifying them visually.

Certainly, there are many differences between the poster medium and the computer medium that might influence how designers perceive the solution space. For example, a typical poster has more display area than a typical computer screen, while a typical computer program is more interactive and more dynamic (i.e., displaying changing information). But it's worth noting that effective design idioms that were perfectly natural in poster designs—such as connecting lines and nonrectangular information elements—largely failed to appear in the computer designs.

TAXONOMY OF LINKS

Our user study's results indicate frequent use of links on paper but not in computer programs. We attribute this difference to the difficulty with which links can be programmed in current UI systems. In order to understand how links can be adapted to visualizations on the computer, we first explore the general diversity of links in information visualizations by proposing a taxonomy for line and link usage (Table 1). At the top level, we distinguish between the use of lines for showing associations and for showing information applicable globally to a visualization, such as the scale of a map. Lines in associative use are what we have termed "links" previously. Links are the

topic of our discussion here.

We propose two main categories of links: links that elaborate on the information being visualized by adding more information which is not previously evident from the visualization; and links that emphasize visual connections already evident from the visualization.

Elaborative Links

There are several types of elaborative links: correspondence links, equivalence links, succession links, grouping links, and links for showing general relationships. Note that grouping links are an exception in that they may not have ends terminating at the objects being grouped.

Emphasis Links

When the connections between objects are already presented but can be enhanced visually, we turn to emphasis links. Alignments of several objects are usually indicated by dashed lines. Distances between objects can be indicated with a ruler stretching from one object to the other. An arrow shows the direction from one object to another without touching the destination.

Note that a link can connect more than two objects to show an n-ary relationship. A link can also connect to another link: Figure 5 shows such a link being used to show the mutual friends of two persons A and B.

We also consider links attached to only one object: these act like tooltips in annotating the objects being pointed to.

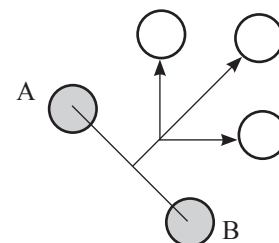


Figure 5. Link showing mutual friends of A and B

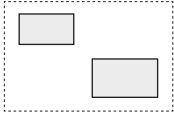
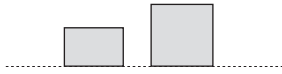
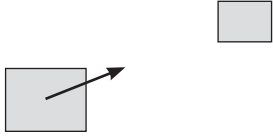
Associative	<i>showing association between different information items</i>	
	Elaborative	<i>the association is not already present in the visualization and is elaborated by the lines</i>
	Correspondence	<i>links between two or more views of a single logical object (e.g., three views of a state are linked together in our redesign example)</i>
	Equivalence	<i>links connecting elements having common or similar attributes, e.g., a contour line indicating points of equal elevation</i>
	Succession	<i>e.g., arrows that show transitions through a flowchart or connect points in a time series</i>
	Grouping	<i>contour enclosing grouped items</i>
		
	General Relationship	<i>links between an object or a relationship to several related objects (e.g., a state is linked to its county data in our redesign example)</i>
	Emphasis	<i>the association is already in the visualization and is emphasized by the lines</i>
	Alignment	<i>e.g., a dashed line on a form designer shows that a UI component being dragged is "snapped" to align with another UI component</i>
		
	Length	<i>e.g., a line indicating some distance between two visual elements</i>
	Direction	<i>e.g., an arrow pointing from one visual element in the direction of another visual element</i>
		
Non-associative	<i>showing information applicable globally to the entire visualization</i>	
	Length	<i>e.g., the scale ruler on a map</i>
	Direction	<i>e.g., the compass on a map</i>

Table 1. Taxonomy of Lines

DESIGN ISSUES FOR RENDERING LINKS

Our initial taxonomy for lines, and links, helps us determine the types of links we will explore through actual implementation. In this section, we dive only into elaborative links as emphasis links are generally more useful in graphics oriented programs rather than in generic information visualizations. We will discuss various challenges to be addressed in order to render links amid the conventional windowing paradigm.

When a link is introduced into the windowing paradigm, different parts of the link are bound to different nodes in the window hierarchy. Since these different nodes lie at different levels in the hierarchy, the link crosses from one part of the hierarchy to another part. Some parts might be obscured by other windows and some parts might be scrolled out of a view port. The challenge lies in how to present a link when only some of its parts are visible. We present two new techniques, *puncturing* and *elasticity*, for handling these problems.

Every link (with the exception of grouping links) consists of one or more *stems* (e.g. straight or curved lines) and two or more *ends* (e.g., arrowheads). Usually, a stem within a link can be partially obscured without detrimental effect to the value of the link. However, when an end of a link is obscured, the value of the link is largely lost. For this reason we shall first consider link end obscurity.

Link End Obscurity Consider the z-ordering of the windows A, B, and C shown in Figure 6a. We considered three choices for showing the link from x to y: not displaying the link at all; clipping the link by the area of B so that the link appears below B (Figure 6b); or ghosting the obscured area of A and the part of the link so that B appears translucent (Figure 6c).

The first option is not desirable because the link would flicker on and off as B is dragged around the space, sometimes obscuring the link and sometimes not. The second option is consistent with the usual clipping policy when the link is considered to be below B itself.

For the third option, translucency may be used in combination with a selection of the details to be rendered. At one end of the spectrum, all details of A and the link show through B. At the other end, only a skeleton of A (e.g., borders of the large components of A) and the link show through. This option may not be desirable if the user is currently focusing solely on interacting with B. We chose to implement the second option in Magpie due to its simplicity. The third option will be explored in the future.

More interesting is the z-ordering in which B intervenes between A and C. We also have three choices: not displaying the

link at all; clipping the obscured end; or showing that the link punctures through B to get to A.

The first option is not desirable for the same reason as in the previous z-ordering. The second option requires the link to be clipped by the borders of the obscuring windows (Figure 7). While this choice seems logical, it suffers from a serious problem: if B is taller, as shown in Figure 8a, then the link does not intersect any of B's four borders and it is not clear where the link should be clipped. We would have the very same problem if we only have the two windows A and C and C itself obscures the link end on A (Figure 8b).

The third option imagines the link as belonging to the third dimension, curving up from C and diving down into B, puncturing through it to get to A (Figure 9a). The part of the link below B is ghosted out when projected onto the screen (Figure 9b).

Link Stem Obscurity We treat the stems of a link to be at the same z-order level as the highest UI element to which the link is attached. This means that any UI element above that highest element will be used to crop the rendering of the link. This is already seen in Figure 6b where B crops the link xy.

Link End Out-of-View When a link end becomes invisible to the user not by obscurity but because it is scrolled out of view, we also face the same problem of how to present the link.

If all ends of a link are cropped through a common view port and some of them are scrolled out of view, we have two options: not displaying the link at all; or clipping the link by the same view port (Figure 10).

When not all ends of a link are cropped by a common view port, we encounter a more interesting challenge. Figure 11 shows that we can crop the stem of the link by a border of the view port that hides the out-of-view end. However, this is not possible if the stem does not intersect any border of the view port, as shown in Figure 12. In order to solve this problem, we propose an "elastic band" view of links: the straight stem of the link behaves like an elastic band and bends at the

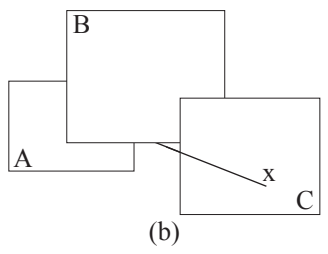
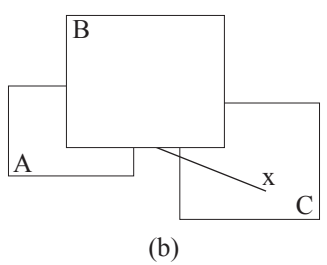
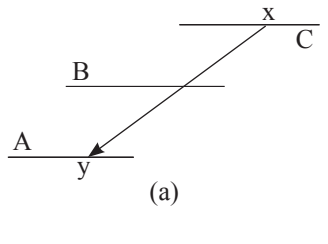
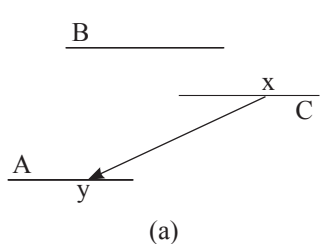


Figure 7. Showing link end obscurity due to intervening sibling by clipping

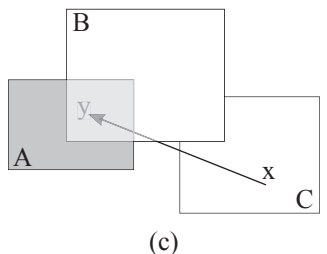


Figure 6. Showing link end obscurity through clipping and ghosting

upper left corner of the view port, as illustrated in Figure 13a and then rendered in Figure 13b.

Link Stem Out-of-View So far we have only focused on the visibility of link ends. There are occasions when stem invisibility creates confusion for the user and renders a link less useful. Consider the scenario in Figure 14a in which two links are cropped by a view port. It is impossible to judge whether the link starting from *a* ends at *c* or *d*; and since all link ends are very close to the upper edge of the scrollable space,

it is not possible to scroll upward to reveal more of the link stems. (Since links are additions to the visualization, the UI component responsible for rendering that visualization is not aware of the links itself and cannot make room for them.) In this scenario, even though all link ends are viewed through the same view port, it is beneficial *not* to crop the link stems by that view port, as shown in Figure 14b.

Another reasonable choice would be to flip the stems upside-down, but when the links are scrolled to the bottom, the stems need to be flipped again. Providing graceful, non-disruptive re-orientation of the link stems is necessary for a smooth user

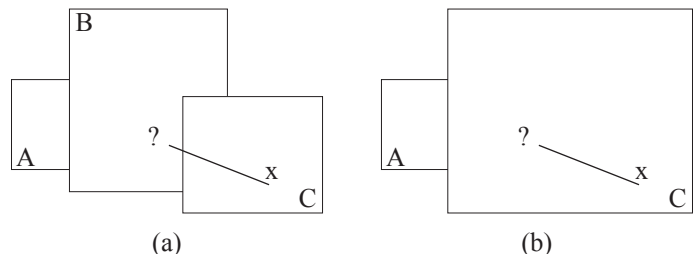


Figure 8. Link end clipping dilemma in presence of intervening sibling

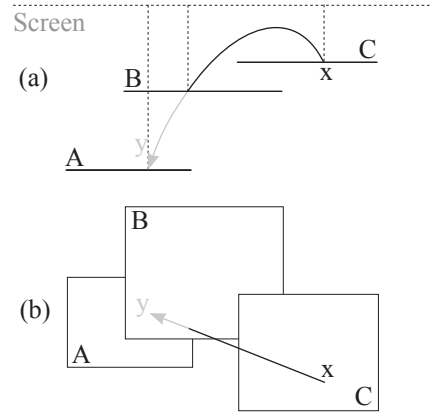


Figure 9. Showing link end obscurity by puncturing

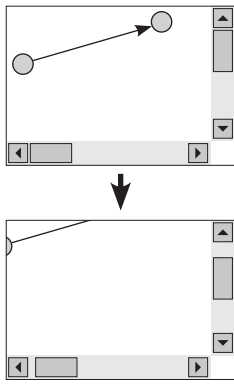


Figure 10. Cropping the link stem when all ends of a link are scrolled out of view

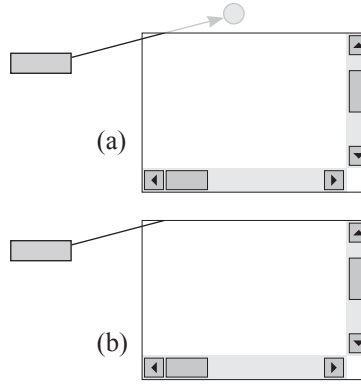


Figure 11. Cropping the link stem when some ends of a link are scrolled out of view

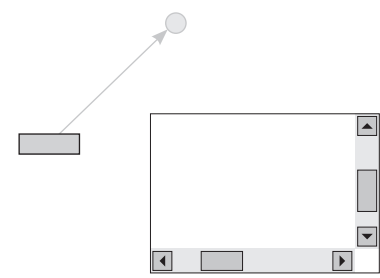


Figure 12. Where to crop the link stem when some ends of a link are scrolled out of view

in which the stem and the ends disappears from the view port. It is useful for keeping the stem visible as long as possible and still manages to hide it away after it is no longer useful to be seen (i.e., its ends are entirely out of view).

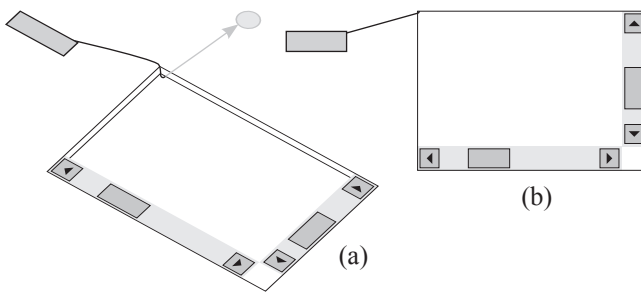


Figure 13. An elastic band link

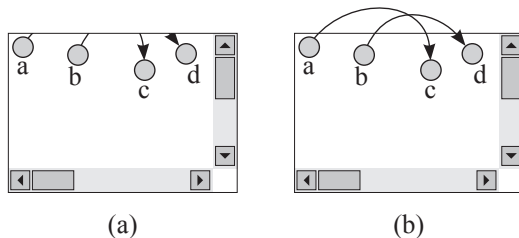


Figure 14. Stem invisibility creates confusion

experience. For that purpose, we propose a physical model of link stems in view ports illustrated in Figure 15. The link stem in the figure is considered to protrude through the view port from the scrollable space. The link stem attaches flexibly to its two ends on the scrollable space; this allows it to bend when it budes against the top border of the view port as the two ends are scrolled upward. Eventually, the stem bends to be almost horizontal and then disappears from under the view port. This physical model yields continuity and coordination

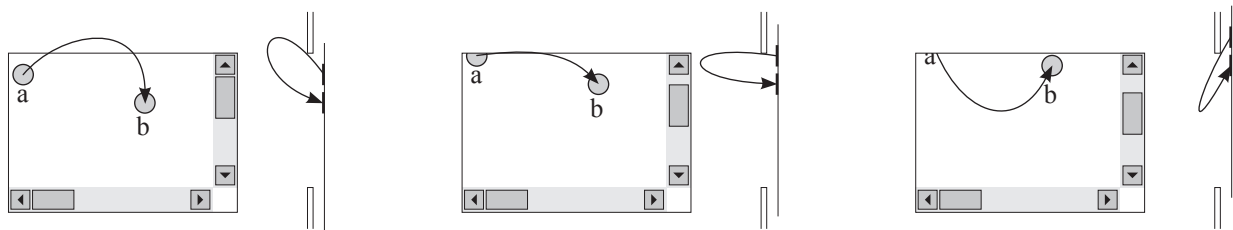


Figure 15. Physical model for link stems protruding from view ports: the link stem is constrained by the view port's upper edge as the view port is scrolled downward

Discussion Throughout our exploration of the design issues for showing links, we have hinted upon many ways in which links seemingly break the windowing hierarchy ever present in today's graphical user interfaces. A link can attach to ends that belong in different subtrees of the windowing hierarchy. A link can protrude out of a view port and defy the view port's clipping. These characteristics of links that made them useful pull them out of the 2½D of the windowing paradigm and demand us to build a 3D or 2¾ model for links.

In presenting links, we have made use of a few uncommon UI mechanisms such as puncturing and elastic band, which incidentally accentuate the 2½D-ness of the UI by exposing relative z-orders of windows and hinting at the layering of a scrollable space with respect to its view port. For example, even if the two windows B and C in Figure 9b are not overlapping, the user can still tell that C is above B because of the puncturing effect.

Conventional UI elements are constrained along the z-order by their parent elements: as an element's parent is pulled up or down the z-order axis, the element itself is pulled along. However, the element is free to determine its X and Y coordinates. The opposite is true for links: a link is pulled along the X and Y axes as its ends move about the screen. However, the link is free to determine its z-order(s). It becomes apparent that links demand layout management not for the two dimensions along the surface of the screen, but for the ½ dimension along the z-order axis, just as conventional UI elements demand layer management for the X and Y dimensions but not for the z-order.

These design issues of links also shine new light on the use of opacity. Previously, a UI element (e.g., a media player window) can volunteer to be translucent. With links, obscured content pushes up through other elements to show itself. Layout management along the z-order concerns not only ordering overlapping elements in depth, but also adjusting their translucency to accommodate how desirable elements are to be seen.

MAGPIE: A TOOLKIT OF LINK UI ELEMENTS

Based on our exploration of various issues on rendering links, we have designed and implemented a toolkit called Magpie for drawing links between different Java/Swing components.

Implementation Technologies

We chose to implement Magpie in Java/Swing as its lightweight components all paint on a common canvas and it is easier to overlay links on them than it is on operating system-native UI widgets. Swing is built on top of the Java 2D Graphics toolkit which offers sophisticated rendering capabilities including alpha composition.

However, at the top level of all Java/Swing UIs are `javax.swing.JFrame`s—operating system-native windows that are independent of one another and do not paint on a common canvas. Consequently, we limit the toolkit to render links spanning only within individual `JFrame`s, not across them. This is a limitation that we will attempt to remove in future versions of the library. Nevertheless, it is still valuable to be able to use such links on, say, `JInternalFrame`s that can be moved about by the user.

Magpie was developed in parallel with our SNAP redesign demo. As such, it was designed to be integrated with an application that uses a `JDesktopPane`. One must be able to incorporate Magpie with minimal alteration to the original program. For this reason, links cannot be Swing components belonging in the same component hierarchy of the original program, as the program's code might make assumptions about its component hierarchy and may be broken due to the intrusion of links. Furthermore, links can assume arbitrary z-order depending on the z-order of their ends and a link can even have several z-orders. Our solution renders links on the `JGlassPane` of the containing `JFrame` and performs all the necessary clipping to give the illusion of appropriate z-orders.

We needed to augment the painting of the `JGlassPane` of the containing `JFrame`, but in Swing, components are rendered by calls to their `paint()` methods and this mechanism disallows augmenting. Consequently, we had to wrap any existing `JGlassPane` with our own. This is a somewhat disruptive intrusion to client applications, but since `JGlassPanes` are not used often, it might be an acceptable solution.

Rendering links ourselves has a few drawbacks. Links are not first class components and much work will be needed to make them behave like first class components, responding to user inputs. A lot of clipping must also be done that can otherwise be handled by the underlying windowing system.

Magpie API

Following are the interfaces of the Magpie library:

- `edu.mit.csail.maggie.IAnchor`: An anchor is something that can be attached to. There are two kinds of anchors: point anchors and shape anchors. For example, a plot can provide point anchors corresponding to the centers of its data points and a map can provide shape anchors corresponding to the boundaries of its regions. When an anchor is moved or reshaped, it fires event to its listeners and its listeners (e.g., links) move appropriately.
- `edu.mit.csail.maggie.IConnectable`: A connectable is a UI element that offers zero or more anchors and, hence, can be connected to. Magpie includes connectable wrappers for the several Swing widgets: `JLabel`, `JList`, and `JTable`. The label wrapper exposes a shape anchor for the label's bounding box, while the list and table wrappers expose shape anchors for the items and selections in the list or table. The custom plot and map widgets in the SNAP example (Figure 16) also implement `IConnectable` to provide their own custom anchors.
- `edu.mit.csail.maggie.IAnchorGroup`: Anchors are offered in groups. A connectable `JList` offers a group of shape anchors corresponding to the boundaries of its selected items. Each anchor group has a name, e.g., “`edu.mit.csail.maggie.selection`”. Anchor groups can be retrieved from a connectable. An anchor group fires events whenever anchors are added to or removed from it.
- `edu.mit.csail.maggie.ILink` is the interface for all links. We have implemented several link classes, each taking a different number of anchors and rendering the link differently. `StraightLineBinaryLink` connects exactly two anchors with a straight line. `StraightLineTertiaryLink` connects three anchors (as shown in Figure 16). `GroupingLink` draws a contour around one or more anchors. Finally, `ToolTipLink` takes exactly one anchor and renders a tooltip connected to that anchor.
- `edu.mit.csail.maggie.ILinkEnd`: Link classes are responsible for rendering link stems only and they make use of `ILinkEnd` classes to render link ends. This design choice allows customization of link ends for each type of link. Magpie currently offers circular link ends (seen attached to the state list in Figure 16), arrow heads (which point at the map and plot in Figure 16), and bare ends (seen in Figure 17).

APPLICATIONS

Figure 16 shows the SNAP redesign demo using the Magpie toolkit. This demo preserves the default windowing look and feel in order to illustrate the improvement that links bring about even in the absence of other alterations proposed in the mockup in Figure 2. Multiple selections are now useful because the user can tell exactly which selection in the plot corresponds to which selection in the list—this is not possible if simple highlighting were used.

We also explored adding Magpie links to an existing Java application, LAPIS [3]. LAPIS is a text editor that uses multiple selections for pattern matching, repetitive editing, and find-and-replace. Magpie links were used to address two known usability problems in the LAPIS user interface. First, LAPIS augments the scrollbar with marks showing where selections are located in the document, so that the user can

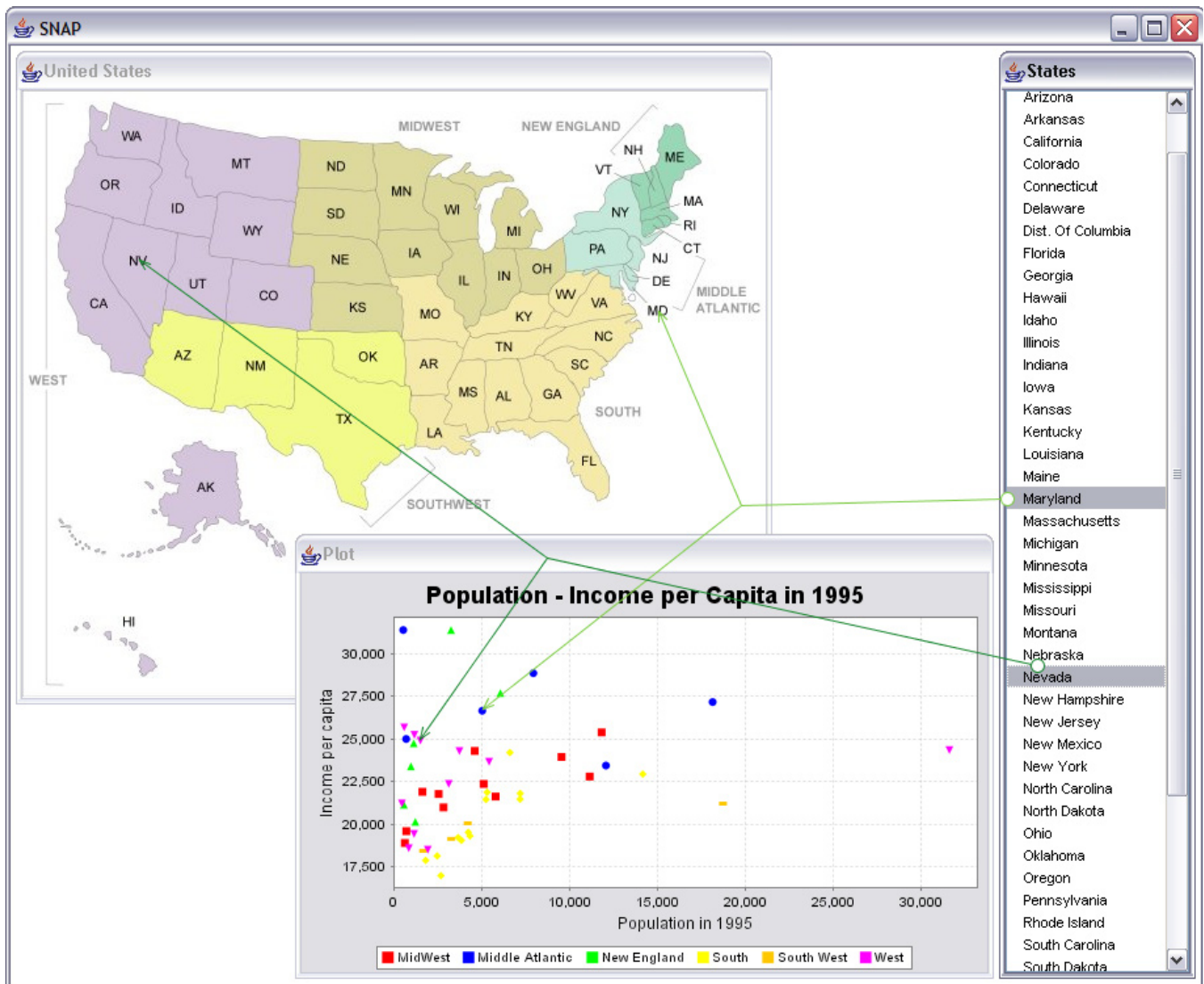


Figure 16. SNAP Redesign demo using the Magpie link toolkit

find them more easily when scrolling around. In user studies, however, new users rarely notice the scrollbar marks or guess their purpose. Magpie links drawn from each scrollbar mark to the corresponding selection in the text makes this connection abundantly clear. (In Figure 17, these links point from the scrollbar to the left, into the text pane.)

Second, we used Magpie links to improve a new feature, cluster-based find & replace [4], which rearranges pattern matches into clusters based on similarity in order to reduce the chance of replacement errors. Clustered matches are shown in a separate pane (on the right in Figure 17), using small snapshots of context around each match. User studies showed that for some tasks, this snapshot provided too little information about a match for the user to decide whether it needed to be replaced. Unfortunately, it was hard for the user to find the corresponding match in the document. Magpie links make this simple: each selected match in the cluster pane is linked to a scrollbar mark, which in turn is linked to a selection in the text pane.

Adding these links to LAPIS required less than 100 lines of new code, which mainly exposes anchors representing scrollbar marks and text selections, and then creates links

between them. Linking to cluster matches was easier, because the cluster pane used a JTree widget already, so anchors for the selections were immediately available after substituting Magpie's JTree wrapper.

RELATED WORK

One of the seven tasks of information visualization is to relate [7]. There are two ways to show associations: synchronizing visual attributes (e.g., color, shape, blinking) and drawing links. The former has been leveraged abundantly. The latter has also been used in numerous work on information visualization. But in most cases where links are used, links are part of the information being visualized, e.g., they are the relationships in a graph. On occasions, links are used to augment the presented information. For example, the Influence Explorer [10] shows histograms of several parameters collected from several experimental subjects and uses links to correspond data points in different histograms collected from a common subject. Augmentation has always taken place inside the same canvas (e.g., a graph view) as the information being visualized. There is one exception, LinkWinds [2], in which links are drawn between different visualizations. However, LinkWinds

limits its links to point only between the linked windows containing the visualizations, not between individual information objects like in our work.

DISCUSSION AND FUTURE WORK

When used within individual visualizations, links are parts of the information being visualized—they show relationships between data objects within the information. In contrast, links between separate visualizations show relationships between the visualizations. For example, in our SNAP redesign example, the list item “Maryland”, the map label “MD”, and the plot data point circle are essentially the same logical object but are presented differently in different visualizations. Hence, the links do not show relations between different data objects but rather reveal the fact that the three visualizations are cooperating to show three different views of the same object.

More generally, links can be used to expose to the user the internal wirings of the UI itself. In this manner, a UI can indicate how information flows through it, e.g., how checking a particular checkbox would change the content of a textbox not so nearby; what object a menu command would act upon.

As presented so far, links are rendered but not directly manipulable by the user. One can imagine allowing the user to reconnect links to request modifications to the information being visualized or to the UI’s internal wirings. The semantics and mechanism for reconnecting links need to be explored.

Like any UI mechanism, links have their niche and their limitations. In particular, it is obvious that links do not scale well: Figure 16 would be incomprehensible if all states were selected at once. However, links are still valuable even with their lack of scalability. When and how links should be used are topics for future work that demand in-depth usability evaluations.

With regards to implementation, we will attempt to incorporate links in a generic constraint-based UI management system because it seems logical to use constraints for tracking link ends. In addition, we will also explore the idea of z-order layout management.

CONCLUSION

In this paper, we propose abstracting links into UI elements that can be used to show connections between other UI elements rather than just within underlying information being presented. In particular, one can show synchronized selections as demonstrated in our SNAP redesign demo, or correspondence between different UI elements as in the LAPIS/Magpie integration.

Through this concept of links, we hint at the need to re-examine the three fundamental characteristics of the windowing paradigm in order to explore deviations from those charac-

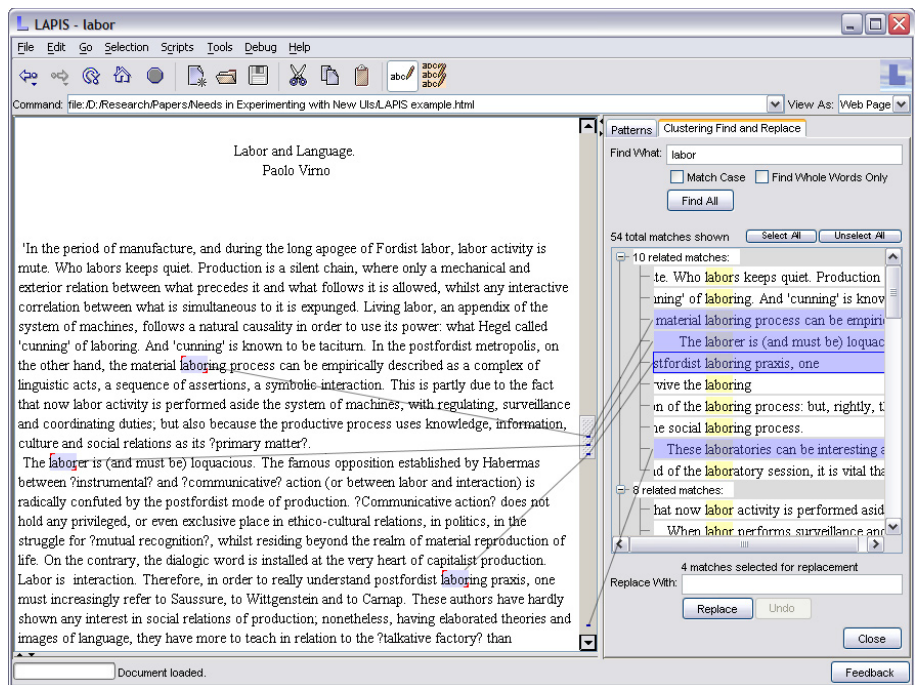


Figure 17. LAPIS with links

teristics for the purpose of increasing expressiveness in UI design.

REFERENCES

- [1] Ishak, E. W. and Feiner, S. K. “Free-Space Transparency: Exposing Hidden Content Through Unimportant Screen Space.” In Conference Supplement of UIST 2003, p.75–76.
- [2] Jacobson, A., Berkin, A., and Orton, M., “LinkWinds: Interactive Scientific Data Analysis and Visualization.” *Communications of the ACM*, 37(4), p.43–52, April 1994.
- [3] Miller, R.C. and Myers, B.A. “Multiple Selections in Smart Text Editing.” In Proc. IUI 2002, p.103–110.
- [4] Miller, R.C. and Marshall, A.M. “Cluster-Based Find & Replace.” In Proc. CHI 2004, to appear.
- [5] North, C. and Shneiderman, B. “Snap-together visualization: a user interface for coordinating visualizations via relational schemata.” In Proc. AVI 2000, p.128–135.
- [6] Paley, W. Bradford. “Designing Better Transparent Overlays by Applying Illustration Techniques and Vision Findings.” In Conference Supplement of UIST 2003, p.57–58.
- [7] Shneiderman, B. “The eyes have it: a task by data type taxonomy for information visualizations.” In Proc. IEEE Symposium on Visual Languages, p.336–343, 1996.
- [8] Stotts, D., Smith, J. M., and Jen, D. “The Vis-a-Vid Transparent Video Facetop.” In Conference Supplement of UIST 2003, p.57–58.
- [9] Tufte, E.R. “The Visual Display of Quantitative Information.” 2nd ed. Graphics Press: Cheshire, CT, 2001.
- [10] Tweedie, L., Spence, B., Dawkes, H., and Su, H. “The Influence Explorer (video) - a tool for design.” In Conference companion on Human factors in computing systems, p.390–391, 1996, Vancouver, Canada.
- [11] Van Dantzich, M., Robbins, D., Horvitz, E. & Czerwinski, M. “Scope: Providing awareness of multiple notifications at a glance.” In Proc. AVI 2002.