User Interface Behaviors for Spatially Overlaid Implicit Structures

Jim Rosenberg
555 Davidson Road
Grindstone, Pa 15442
E-mail: jr@amanue.com

ABSTRACT
Spatial hypertext systems typically assume that when objects are placed in proximity as a spatial aggregation, at least some part of each object will be visible, allowing an object to be selected by clicking on a visible part. This presents problems when an object becomes completely occluded. This paper presents various user interface behaviors which seek to solve the problem of discovery and exposure of completely occluded objects. Behaviors are divided into current behaviors, discovery behaviors, and exposure behaviors. Finally general issues are discussed, such as screen geometry requirements, pluggability, and negotiation among objects for user interface resources.

INTRODUCTION
Spatial hypertext systems, such as VIKI [4], CAOS [5], and VKB [8] provide facilities for implicit or emergent structure by supporting spatial proximity as a form of association. The most intimate form of proximity is an *overlay*, in which items are placed on top of one another. Nevertheless, existing spatial hypertext systems provide only very weak support for overlays, since they assume that some part of each element in a spatial aggregate is always *visible* where it can be selected by clicking with the mouse. The problem, in a nutshell, is that when elements are overlaid an element can become completely occluded. Some explicit form of user interface behavior is necessary so that such elements can be discovered and accessed. This paper presents several such candidate behaviors. Most of these behaviors have in fact not been implemented; they are being presented here in the hope that spatial hypertext system implementers will find their ideas fruitful for incorporating into future systems.

In the examples that follow, figures illustrating user interface behaviors all refer to the same "baseline" spatial overlay, as shown in Figure 1. Figure 1 shows three VKB-like collections overlaid; the titles of two of them ("Anti-Conjunctive Drift" and "Sub-screening" are visible, while a tiny sliver from the third collection is just barely visible behind the other two. Within each collection are individual objects which

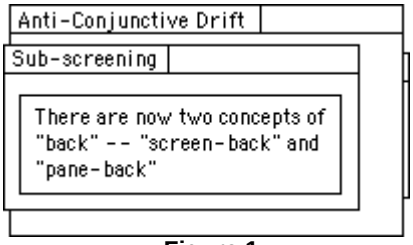

Figure 1
Three overlaid collections are shown, with only a sliver of the third visible at right. Each collection has two or more objects. All other figures in this paper refer to this overlay, with various behaviors to discover and show the objects.

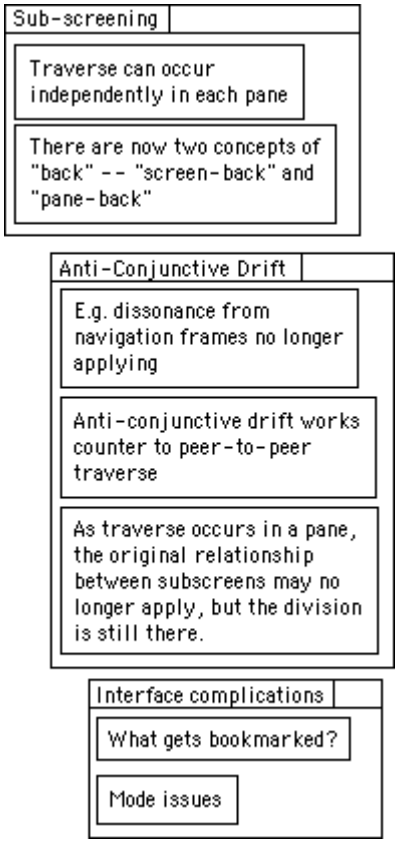

Figure 2
The objects from Figure 1 spread apart

are also heavily overlaid, so in fact there are 3 collections and 7 individual objects represented. Figure 1 shows only a single individual object readably; it is exactly how to discover and display the hidden objects that is the subject of this paper. Figure 2 shows the objects spread apart so that each one is visible. (Spreading is an important specific behavior, which will be discussed below.)

CURRENT BEHAVIORS

Individual Selection

All current spatial hypertext systems allow an object to be selected by clicking on some visible portion of it. Clearly this leaves completely unsolved the problem of selecting an object which is completely occluded. VKB allows selection of an occluded object by means of an *object hierarchy view* which is separate from the spatial view. This paper addresses methods by which occluded objects may be selected within a spatial view.

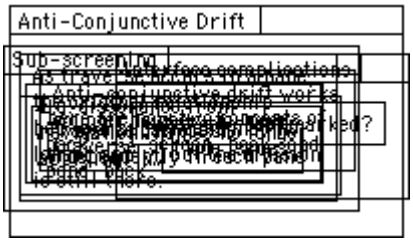

Figure 3 — Transparency
All objects from Figure 1 rendered transparent

Transparency

VKB allows the user to make objects transparent. Figure 3 shows the result of making all objects in Figure 1 transparent. As is readily seen, the result may be artistically interesting from a visual point of view, but does not contribute to the problem of navigating to and selecting specific objects.

DISCOVERY BEHAVIORS

Discovery behaviors alter the appearance of the visual workspace so that the location of hidden objects may be revealed; typically these behaviors do not show the entire contents of the objects revealed.

The Frame X-ray

When the Frame X-ray behavior is invoked, a section of the visual workspace is selected and all objects in that section are rendered transparently, with borders intact but no "content". Figure 4 shows the results of the Frame X-ray applied to Figure 1. While the objects in Figure 3 are almost indiscernible one from another, it is surprisingly easy to pick out individual objects in Figure 4. Note in particular that the hindmost collection is quite easy to spot. The idea of the Frame X-ray display is that object borders are available for selection even on objects that are many layers down.

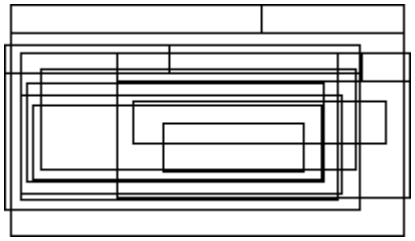

Figure 4 — Frame X-ray
Objects from Figure 1 rendered transparently but with only their borders showing

While the Frame X-ray concept is very simple to understand and should be fairly easy to implement at the application level, it does have some clear drawbacks. (1) While the existence of objects is made clear, they are not differentiated one from another by content. (Though border color distinctions would be available.) (2) The behavior it introduces is likely to be highly modal. An object whose outline is revealed in a Frame X-ray display must be selectable. I.e. interactivity of the visual workspace must be preserved with some degree of orthogonality with respect to Frame X-ray display vs. normal display. This complicates the user interface; designers may have strong opinions about modality. (3) The Frame X-ray display itself is confusing if objects are highly aligned geometrically.

The Core-Drill

The core-drill is a moveable geometric region of the screen — presumably a rectangle — which has popped up next to it a display of names of all the objects it intersects. This is illustrated in Figure 5. When an object from the menu is selected, it is brought to the top; if that object is part of a collection then presumably the whole collection is brought to the top. Note the menu in Figure 5 is shown as "smart" about collections: objects inside a collection are shown indented under the collection name.

The core-drill allows a very effective form of navigation of the visual space, in spite of a great density of overlays; if the associated pop-up menu has scroll bars, then there is no limit to the amount of overlaying that can be navigated. However, there is one obvious drawback to the core-drill: to be effective, all objects must have names. This is a very major issue. Spatial hypertext was created in the first place to avoid the problem of premature commitment to structure [3]; premature commitment to *naming* is not all that different from premature commitment to structure. Figure 5 shows some names that were "manually" created. Of course an *agent* could create names for objects dy-

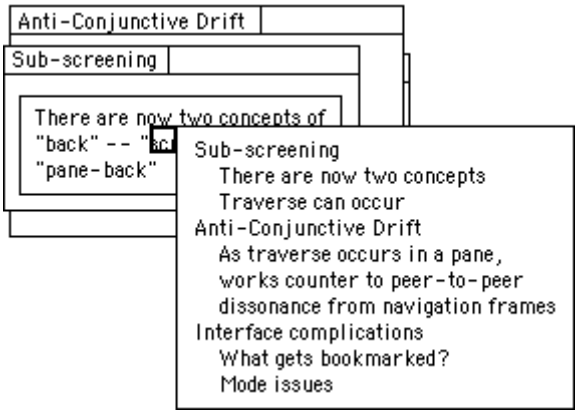

Figure 5 — The Core-Drill
The small rectangle is moveable on the screen, and causes a pop-up menu of all intersected objects to appear. The menu is shown with "smart indentation" for collection membership.

namically — which might work well if all objects are text objects — but there remains the problem of how names would be created for non-text objects, such as graphic images or multimedia objects.

The core-drill also has modal issues; presumably there is an interface behavior or keyboard shortcut that causes it to appear and disappear, and modality of dragging around or resizing the core-drill vs. "normal" behavior of the spatial workspace could be tricky.

The Glass-Bottom Boat

The Glass-Bottom Boat is a special window which "slices through" the layering of objects on the visual workspace to reveal whatever may be present several layers underneath the surface. This window has explicit gadgets to raise it or lower it one layer, and an indication / setting to show how many layers down from the surface of the workspace it is. The Glass-Bottom Boat is quite attractive from a modality point of view: presumably it acts just like any other window, and within the sub-layer it reveals in its contents, events are passed through exactly as if that layer was on top. I.e. user interface behavior inside the Glass-Bottom Boat window is identical to behavior were the same contents to be part of the top layer. In this sense the Glass-Bottom Boat is not modal, and uses no interface resources that might be desired for some other purpose. Alas, this very power reveals the main weakness of this concept: in order to be implemented properly it might have to be located in the native operating system windowing system itself. Commercially viable windowing systems are not normally accessible to such interventions. For windowing systems implemented at the application level (e.g. Smalltalk) it might be feasible, however. The Glass-Bottom Boat is illustrated in Figure 6.

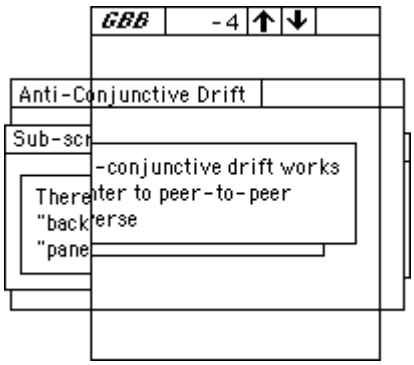

Figure 6 — The Glass-Bottom Boat
The "GBB" window slices through the layering of the visual workspace to reveal a view 4 layers deep. Within this window the interface works as it would were the revealed layer on top. E.g. the object underneath the "conjunctive drift works" object could be selected just by clicking.

In spite of the simplicity of this concept, there are likely to be some surprises if it is actually implemented. Consider a Glass-Bottom Boat window that is fairly large, and which is several layers down, as shown in Figure 6. Suppose an object is selected from inside the Glass-Bottom Boat. To what layer should the Glass Bottom Boat window move? If (as discussed above) the interface inside the Glass-Bottom Boat works "identically" to the normal interface, the selected object moves to "the top". What is the top? Is it the top with respect to the current position of the Glass-Bottom Boat, or the top of the entire windowing system? If the object moves to the top of the entire windowing system, then we have the paradox that it might disappear from the Glass-Bottom Boat! Should the layer of the Glass-Bottom Boat then follow the selected object? Should the Glass-Bottom Boat automatically close? User preferences may need to be consulted to answer such questions.

EXPOSURE BEHAVIORS

Whereas the discovery behaviors discussed above tend to simply identify where a possibly occluded element in an overlay may be found, exposure behaviors tend to expose the entirety of an element. (Actually the Glass-Bottom Boat is a hybrid that could be classified either as an exposure behavior or a discovery behavior.)

The Frame Stack

When the mouse approaches an overlay having this behavior, a stack of frames is displayed, one for each element of the overlay. These frames serve as "On MouseOver" hot-spots to allow navigation of each member of the overlay; note that the frame stack explicitly embeds the *peer structure* of the overlay. The frame stack is illustrated in Figure 7. The frame stack

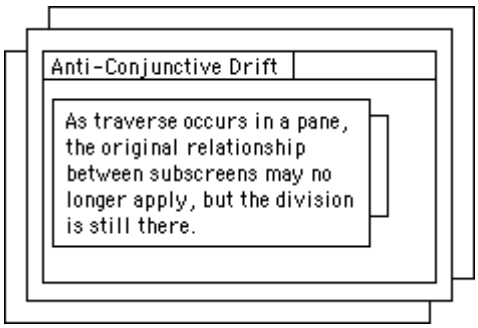

Figure 7 — The Frame Stack
This shows the frame stack "opened" to the middle collection. The frames (rectangles) act as "On Mouse-Over" hot-spots: as the cursor moves to one of the other frames in the stack, that element moves to the top.

concept has been implemented in specific literary works [6], though no authoring environments make it easy to implement as an "off-the-shelf" behavior. The frame stack concept works well with nested collections, since it allows a frame stack to be opened inside an outer frame stack. One drawback to the frame stack concept is that it is quite expensive in terms of screen real estate. Assuming all frames in one stack are the same size, this must be large enough to accommodate the largest element — with sufficient room for a bit of margin. Note that as rendered here, if a frame stack were to be opened on the contents of the collection shown on top in Figure 7, there would not be enough room to fit all the frames inside the current collection boundary.

The Spread

This behavior was explored by Mander et al [2]; the spread simply spreads out members of a pile so that all of them are visible. Figure 2 shows a "recursive spread" of the example pile used throughout this paper. Spreading uses a technique which may be called *co-presentation*, which is discussed in detail in [7]. While the simplicity of spreading is appealing, there are a number of issues with this behavior. It appears that for the foreseeable future, screen real estate will always be in short supply; what happens when a pile needs to be spread out over more real estate than is available? In this case we will still have members of the pile potentially occluding one another, which means some other behavior will be needed. When piles are *nested*, presumably each collection would have to occupy a disjoint space (as in Figure 2) with spreading inside that space. This explodes the space requirements significantly.

The Viewing Cone

Mander et al presented another idea for navigating piles, which they termed a *viewing cone*. Their implementation of piles showed a collapsed pile as a 3-dimensional stack so that an edge from every member of the pile is visible. The viewing cone expands a particular member of the pile to a thumbnail or larger view. Again, it is not clear how this would work with nested piles, and also has the problem that at least some part of each member of the pile must be visible; the viewing cone cannot locate an object if it is completely occluded.

Geometric Rectification

There are many ways this behavior can be implemented; it is really a special form of spread, except that parts of objects may still be overlaid. At least one form of this behavior is widely available: many multi-window GUI applications have a "cascade" option for viewing windows, that arrays the windows so that their title bars are all visible as a descending "slant". This behavior has many of the issues of spreading.

GENERAL BEHAVIOR ISSUES

There are several general issues that pertain to all of these behaviors.

Screen Geometry Requirements

In a system like VKB, a collection is assumed to have a particular size, which is a property of the collection; it can also be *zoomed* to occupy the full available screen area. It is clear that several of the behaviors discussed above may have additional conditional requirements for screen real estate. Spreading requires enough real estate for all items to become exposed; the frame stack requires enough real estate to show a frame for each element of a pile, etc. How does an object such as a pile communicate to the user interface system how much real estate it requires for some behavior which will only occur when the user initiates some particular action? How does the user interface system communicate to an object how much screen real estate is available?

Pluggability

Ideally, user interface behaviors of the kind discussed here should be pluggable; as new behaviors are invented it would be wonderful if they can be implemented in existing systems without having to completely reimplement a system. Of course this requires a *framework*, which is a significant effort. The Model-View-Controller Paradigm [1] represents a classical effort to implement pluggable user interface behaviors; it is complex and in spite of frequent revivals often not used.

Spatial hypertext systems may have special issues with regard to pluggable behaviors. The whole rationale of spatial hypertext is based on supporting emergent and ambiguous structure. An object may be placed "near" another object to indicate an ambiguous or not-yet-de-

fined relationship to that object; this association might not be represented by any persistent structure, but might be computed on the fly, based on criteria that can change depending on what the user does. To the extent that user interface behaviors have geometry requirements, such requirements may have interactions with such on-the-fly computations. How do these communicate? For example: computation of available real estate may be affected by what counts as "near" in spatial parsing algorithms. If a spread is supposed to avoid overlaying a nearby collection, the available real estate for the spread is constrained not only by the actual real estate occupied by the nearby collection, but by a surrounding area which the spatial parser would "count" as part of the collection. Thus a spatial parser would have to respond not only to requests to indicate what objects are "in" an aggregation, but also what *areas* include objects that *would be* considered "in" an aggregation just by being present in that area.

Behavior Negotiation

A common theme of user interface behaviors is that space requirements can vary depending on the state of an object. An object which is under active investigation may be "expanded", and when not receiving attention may be "collapsed". Current spatial hypertext systems do not support *negotiation* of objects for screen real estate. For instance, in VKB it is assumed that when a collection needs attention, it can be expanded to fill the whole screen — eliminating from view "sibling" collections. A system of negotiation would allow an object that needs more screen real estate to obtain it by a variety of means. If expanding the collection to the full bounding box of all objects in that collection does not collide with the space of any other object, it could be expanded to just that amount of space, with no impact on other objects. If not enough real estate is available, nearby objects could be "asked" to collapse. Finally, if insufficient real estate is still not available, objects could be overlaid. It goes without saying that the design of a protocol whereby such negotiation would occur is a significant undertaking.

The concept of spatial hypertext includes the idea that certain objects need to be seen "together" — if at all possible. This calls for some subtlety in the design of the user interface when these same objects (alas) *compete* for user interface resources. The requirement of objects for "coattention" has unexplored consequences for user interfaces.

REFERENCES

1. Krasner, G. E., and Pope, S. T., "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", *Journal of Object Oriented Programming*, August/September, 1988, 26-49.

2. Mander, Salomon, Richard, Gitta, and Wong, Yin Yin, "A 'Pile' Metaphor for Supporting Casual Organization of Information", *CHI '92*, ACM, New York, 1992, pp. 627-634.

3. Marshall, Catherine C. and Rogers, Russell A., "Two Years before the Mist: Experiences with Aquanet", *ECHT '92 Proceeding of the ACM Conference on Hypertext*, ACM, New York, 1992.

4. Marshall, Catherine C., Shipman, Frank M. III, and Coombs, James H., "VIKI: Spatial Hypertext Supporting Emergent Structure", *European Conference on Hypermedia Technology 1994 Proceedings*, ACM, New York, 1994, pp. 13-23.

5. Reinert, Olav, Bucka-Lassen, Dirk, Pedersen, Claus Aagard, and Nürnberg, Peter J., "CAOS, A Collaborative and Open Spatial Structure Service Component with Incremental Spatial Parsing", *Hypertext 99: The Proceedings of the Tenth ACM Conference on Hypertext and Hypermedia*, ACM, New York, 1999, pp. 49-50.

6. Rosenberg, Jim, *The Barrier Frames: Finality crystal shunt curl chant quickening giveaway stare*, Eastgate Systems, Watertown, 1996, sample http://www.well.com/user/jer/inter_works.html#Barrier_frames.

7. Rosenberg, Jim, "And And: Conjunctive Hypertext and the Structure Acteme Juncture", *Hypertext '01: Proceedings of the 2001 ACM Conference on Hypertext*, ACM, New York, 2001.

8. Shipman, Frank M. III, Hsieh, Haowei, Maloor, Preetam, and Moore, J. Michael, "The Visual Knowledge Builder: A Second Generation Spatial Hypertext", *Hypertext '01: Proceedings of the 2001 ACM Conference on Hypertext*, ACM, New York, 2001.