Navigating Complex Information with the ZTree

Lyn Bartram
School of Computing Science
Simon Fraser University
Burnaby, BC, V5A 1S6
CANADA
lyn@cs.sfu.ca

Axel Uhl
ABB Corporate Research[†]
Heidelberg
GERMANY
axel.uhl@io-software.com

Tom Calvert
Technical University of British Columbia
Surrey, BC, V3R 7P8
CANADA
calvert@techbc.ca

[†] Current address: InterActive Software Objects GMBH, Freiburg, GERMANY

Abstract

This paper discusses navigation issues in large-scale databases and proposes hypermap visualizations as effective navigational views. We describe the `ZTree`, a technique that allows users to explore both hierarchical and relational aspects of the information space. The `ZTree` uses a fisheye map layout that aids the user in current navigational decisions and provides a history of previous information retrieval paths.

Key words: Information visualization, interactive techniques.

1 Introduction

Complex information systems demand different interface approaches from the usual desktop paradigm. People increasingly get lost in electronic space. Navigation is a familiar activity in the real world but a bewildering process in abstract information spaces which lack analogous cues and tools to situate and guide the user along desired routes. The challenge facing us is how to facilitate navigation in such systems without imposing extra cognitive overhead.

Complexity in these spaces is a function of size, scope and organisation. One taxonomy defines information structures as anarchic (arbitrary organization) vs. moderated (imposed organization) and known (fully defined) vs. unknown (constantly changing) [16]. However, even well-structured and fully-defined spaces can be "unknowable". In many applications it is the relationships between elements of information that are as interesting as the information itself, and these relationships may be both fixed (imposed by the information model) and dynamic (created by the user as part of the information assimilation task). Further, in many complex spaces it is the user's path through them (the trace of the information retrieval "dialogue") that is of interest rather than individual nodes. We use Tweedie's concept of *derived* [24] rather than *fully known* structure to express this accumulated set of retrieval paths. A user may derive different structures from the same underlying space in the context of different searches and problem-solving tasks.

While much work in information navigation has been in the well-known domain of the World Wide Web ([6,9,10]), many other information spaces present equally challenging problems. The project described in this paper concerns large-scale databases for engineering applications in which the information submits to a defined structure but is of such a size and scope that it cannot be fully known by any one user. Simple query-based approaches are insufficient: users need to understand the information in given contexts and quickly find relevant relationships to other such contexts. We are interested in discovering which elements of information visualization techniques apply well to navigational support in such systems.

Navigational approaches in large information space visualization tend to fall into two categories. Multiscale interfaces such as Pad++ [5] and EPS [7] operate on the information space directly and allow variable levels of exploration. More common are *structure views:* abstractions of the underlying structure which provide the user with selectable points to deliver detailed information in another display area (such as file system views). Our research focuses on the use of explicit map-based representations as structure views.

This paper describes the `ZTree`, a fisheye view based on the Continuous Zoom [3] which provides a 2D navigational map. A major issue in applying such techniques as mapping tools is the layout of the structure. An automatic layout algorithm was developed to accommodate dynamic structure accumulated as the user moves through the information space.

The paper is organized as follows. We discuss general issues of navigation and define the *hypermap*, an effective navigational tool for complex spaces. A overview of related work follows. We then briefly describe a large-scale engineering logistics application (SiLog) and the issues in navigating the information space as context for the description of the `ZTree`

interface based on the CZ and the `Grid` automatic layout algorithm. We conclude the paper with some open research questions and suggestions for further investigation and development.

2 From Structure Views To Hypermaps

How do people find information in a complex space? Furnas defines navigation as "moving oneself sequentially around an environment, deciding where to go next based on the task and the parts of the environment seen so far" [16]. Effective view navigation requires small views and short and discoverable paths [13]. Users need support in constructing an overview of the information space [16]. This helps them construct a mental map of the information structure and topology. In addition, they need history: where they've been and the set of connections and traverses they have made. A study of WWW navigation found that visualizing the entire structure was less important than being able to re-find interesting pages one had previously visited and retrace associations [10]. Finally, because navigation is a cognitive activity that requires context, users need a persistent representation of the overall space.

Humans rely on maps to navigate in the physical world. We believe that graphical maps of the information structure which elicit the familiar process of map-based navigation will be most useful in our complex environment. We extend the definition of a GIS *hypermap* [18] to be any 2D or 3D structure view which evokes three key aspects of paper-based maps: persistent overview of navigation possibilities; identification and retrieval of key features; and identification of how features relate to one another.

A 2D fisheye map of a WWW space was found to be useful because it evoked two key characteristics of paper-based maps: identification of key features, and identification of relationships between those features [10]. Map layouts exploit human spatial cognition abilities, and it appears that substantial distortion can be applied as long as relative positional constraints are respected [11].

3 Related Work

Many approaches use tree models to organize information space hierarchically and variants of tree structures to display them. TreeMaps [2] used a 2D space-filling approach to lay out a tree-structure, in which higher nodes in the tree got more space in the tree map. The resulting display gave effective contextual information and elicited the immediate perception of distribution in the tree, but individual features were difficult to discriminate once the tree was reasonably populated. Cone Trees [21] are a 3D extension to traditional 2D tree layouts, allowing a much greater amount of information to be concurrently shown at the cost of some occlusion. The user may have to rotate a node to get access to a subtree, but smooth animation greatly reduces the cognitive transition of re-orienting the tree. Several database visualization systems incorporate Cone Tree approaches. WINONA models class-object structure in a Cone Tree view [20]. LyberWorld, a hypertext document database, uses Cone trees to represent the information retrieval history by building up a view of query paths [14]. Of particular interest is PadPrints [5], a 2D multiscale hypermap of a user's path through the WWW based on Pad++[6]. PadPrints dynamically builds up a tree structure of nodes corresponding to pages accessed which allows the user to maintain as much context as desired (by zooming in and out of the structure view) while viewing detail in the standard browser window.

All of the above are constrained by an inherent limitation of tree views: relational information that does not correspond to hierarchical structure cannot be easily shown. Thus there is great interest in exploring the flexibility and extensibility of graph visualizations. Both OFDAV [9] and NicheWorks [25] are examples of graph layouts for WWW navigation that manage arbitrarily large graphs. OFDAV does so by only rendering a sub-graph around a current focus node and giving cues that lead "off-screen": the user never sees the entire context. NicheWorks employs specialized layout algorithms to cluster related elements in a graph closely together in perceptual clumps, highlighting only a few elements of interest. Design Gallery Browser [1] uses 2D and 3D graphs to layout semantically organized clusters of similar images. Thumbnails of the images surround the graph and are connected to their relevant node by links. Users navigate to a cluster by panning and zooming in the space to select full size image views, so overall context can be lost.

The lack of hierarchical structure in such graph-based approaches makes them harder to navigate than trees. Moreover, moving around the graph and adding nodes results in often disorienting reconfiguration of the layout which seriously perturbs any perceptual map the user may have had of the space, complicating feature retrieval.

A detail-in-context approach which models both hierarchical and relational structure is the Continuous Zoom [3] (CZ). Parts of the information space are summarized by being contained in closed cluster nodes, while the user can open other clusters to successively examine finer levels of detail. Such fisheye approaches[12] have been generally shown to have advantages whenever users find themselves in an information space which is too complex to easily render on one small display or window [23]. However, they tend to suffer from problems of distortion [15,22]. Our intuition is that such techniques may hold more potential as hypermaps than for their previously explored applications of direct information visualization.

Most CZ applications have been to direct

visualization of the underlying information space in which detail was mapped onto the leaf nodes. The resulting screen space issues limited the technique. However, recent approaches which use a CZ basis as structure views for network management and the WWW [10,11] have proven successful. A sample visual programming application built with the Hyper Mochi interface [27] (a CZ-based technique) suggests that the combination of hyperlinks and hierarchy provides an intuitive navigational model.

4 The Silog Project: Large-scale Logistics

Building turnkey power generation installations is an enormously complex logistical undertaking, requiring the coordination of hundreds of suppliers, thousands of shipments and millions of parts within tight time and quality constraints. An important aspect is the flow of information between all participants along the supply chain. The ABB *SiLog* (Site Logistics) application is designed to streamline processes on-site. These include handling on-site material delivery and reception, purchase requests, providing feedback on missing and broken parts and communicating changes in due dates.

Visualization and navigation of the complex logistical data must support different information retrieval and management strategies for people in different roles. For example, the project manager puts in a request like "by when do we have all parts for assembly group #123?", whereas the material handling manager might ask "where in our outdoor storage is the crate with ID #987?". Role needs are modeled as *use cases*, each with differing requirements for how the data is visualized, grouped and arranged. Not only is there an enormous amount of object data, there are also a variety of relationships between the different entities like delivery items, purchase orders or transport items. A lot of them are *one-to-many* relations where, for example, a purchase order knows the set of all delivery items it subsumes. At the same time, each delivery item pertains to a particular shipping bill of materials, which, in turn, belongs to a purchase order. Thus entities can be reached in a variety of ways. This makes visualizing the available information and streamlining the navigation a challenging task for two reasons. First, the scope and size of the information space exceeds the scope of any individual user (complexity), so the space is essentially "unknowable". Second, users need to see only the subset of the information in context to their roles. The standard query-filter-requery approach provides detail but quickly strands them in space. Thus they need a graph of both entity and structural information; more precisely, they need the derived structure [24] of the information space that fits their roles and expands to accommodate information retrieval activities.

Because large graphs are inherently difficult for humans to navigate [13,25], SiLog maps the object graph onto a hierarchy, interpreting certain edges as a containment relation. Hierarchical views tailored for particular use cases are defined for the object graph. The hierarchies are arranged so that the user gets a view that corresponds with the use case's terminology and logic. This lets us present the graph as a familiar tree view and gives the user a simplified representation of the space. Tree views allow the user to drill down and roll up specific branches of the tree (by expanding nodes and collapsing subtrees) and are generally considered to be an effective navigational tool when the underlying structure is moderately balanced [13]. However, the tree model is problematic because the structure is not exclusively hierarchical, but is also a graph with links that are not modeled by containment in the hierarchy.

To handle this problem SiLog models *canonical paths*. Each object contained in a hierarchy view has exactly one canonical path associated with it leading from the root of the hierarchy to that object. Other non-canonical paths may exist in the hierarchy leading to the same object. This conceptualizes the fact that there is a way to reach an object that is more usual to a user in a given context than any other way, but that there are other potentially relevant paths to that information. We use this concept to handle links that run across the hierarchy. Tracing a selected node in the tree view back to the root of the tree results in the navigation path leading to that node. Whenever a node in the hierarchy offers a link to an object whose canonical path does *not* start with the current navigation path, traversing this link will expand the canonical path for the reached object in the tree view and display the object there. This way, each object is represented at most once in the whole tree, but can be related to and accessed from many other parts of the tree.

While this approach provides a more tractable model of the underlying information space, it is problematic at the interface. The original SiLog interface comprises two views: a detailed *content view*, usually tables and lists of data at each node, and a tree-based structure view. The structure view supports "coarse" navigation to the node required: the user can see appropriate attributes and details in the `content view`. Selecting items in the `content view` is analogous to exploring a relationship (a cross-hierarchy link) and may result in new paths being opened in the structure view. Figure 1 shows the initial design of the structure view: an indented scrolling list akin to familiar file system viewers, called the `JTree` after the Java widget used to implement it.

The `JTree` is inadequate for visualizing and navigating even small prototype projects. The list rapidly gets large and is awkward as soon as the user needs to manipulate or discover entries that may have scrolled off the page. The user quickly loses context. Moreover, there is no clear way to emphasize or even detect relations between entities that are not hierarchical. In the prototype of Figure 1, the user

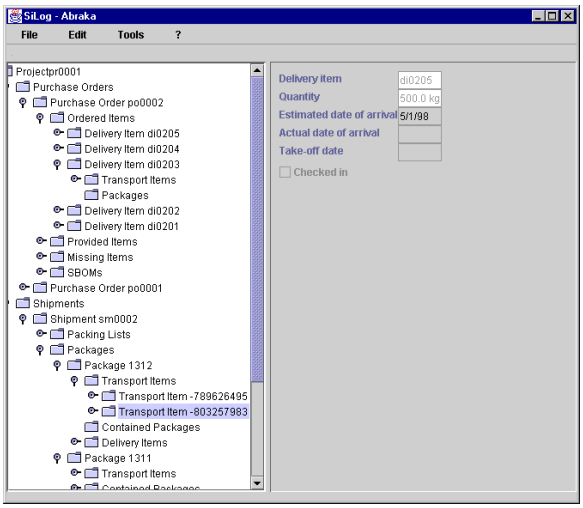


Figure 1. The `Jtree` SiLog interface

explored the **Shipments** subtree to reach **Transport Item 803257983.** Selecting a delivery item from the accompanying `content view` resulted in an abrupt change to the structure view: a path was opened in the **Purchase Order** subtree to the related element and the focus "jumped" to it with no accompanying visual cue to indicate the relationship and explain the sudden change to the structure view.

Our goal was the design of better navigation support for the SiLog application by combining the power of hypermapping techniques with the flexibility of detail-in-context offered by multiscale approaches. We identified the following criteria for a SiLog structure view.

- Effective view traversal and navigation: users didn't want to expend too much effort tracking down information.
- Preservation of context: information can be reached by several paths.
- Explicit visualization of relations between information elements. Derived structure must involve both paths (tree descents) and relationships (cross-hierarchy traverses).
- Automatic layout. Users do not need to see the entire space but only the subset of interest. Thus the structure view gets populated "on the fly" by user queries and needs to be dynamically reconfigured in such a way that the transitions are easily followed. Since the user's task is to find information and not to reorganize the view, this reconfiguration should not require user intervention.

Because it supports simultaneous detail and contextual views, hierarchical and associative structure, and spatial cognition [11], these criteria led us to select the Continuous Zoom as a basis for a hypermap.

5 The `Ztree`: A Continuous Zoom Hypermap

CZ models a hierarchical data structure with added links between arbitrary nodes in the hierarchy. Nodes represent discrete points in the information space: links represent relationships. Therefore CZ can effectively represent combinations of trees and graphs. Parts of the information space are summarized by being contained in closed *cluster* nodes while other clusters can be opened to successively examine finer levels of detail.

As described in [3], CZ manages a rectangular 2-D display space by recursively breaking it up into smaller rectangular areas, creating a hierarchy of nested rectangles. The user controls the amount of detail in different areas of the display by opening and closing clusters. The contents of an open cluster are visible, allowing one to see the deeper (more detailed) levels of the hierarchy. Closing a cluster effectively prunes a portion of the tree from the display, reducing the detail shown for that part of the system. Open clusters are allocated more space than closed clusters. In addition to this automatic resizing of cluster nodes, the user can enlarge or reduce any node on the display. Through opening and closing clusters, and resizing nodes, the user has complete control over the amount of detail seen in each part of the display. Since the entire hierarchy is visible at all times, the detailed portions always appear in context. Multiple areas can be zoomed simultaneously.

Our previous experience with the CZ in network visualization [4] indicated it was effective for navigating large, hierarchically structured graphs. The CZ effectively supports both the hierarchical and associative (topological) thinking which are essential components of information searching [17,19]. Another reason is the explicit support the CZ provides the user for recognizing and understanding her present location in the information space, a feature targeted as a major need in other complex, multiply-linked information spaces such as hypertext [26]. Finally, CZ layouts appear to exploit the spatial cognition aspects of graphical maps [11] without suffering from the excessive distortion drawbacks of other 2D fisheyes. We have hypothesized two reasons for this. First, CZ layouts do not violate relative layouts: that is, essential "left of", "inside", "on top of" relationships which are fundamental to cognitive consistency [11]. Second, smooth animations perceptually guide the user through the view transition.

5.1 `ZTree` Description

The `ZTree` is a general-purpose widget responsible for specifying an initial CZ layout, responding to application-specific events and defining how the CZ view controller interacts with other application views (in the SiLog application, the `content view`.) It renders some subset of the data model up to and including the entire data model based on the user's and/

or programmer's specification. When the information space is too large, application-defined heuristics can control what subset of the space is initially presented. In the SiLog case, the initial structure view only contains nodes with less than 10 children to reduce the starting complexity. This prunes large branches of the tree. Users build up a richer derived structure view through subsequent queries and browsing in the detailed view.

When the SiLog application is launched two windows appear: one with the standard SiLog view, and one with the initial `ZTree` view. Using the `ZTree` is somewhat similar to using the `JTree`. The user can open and collapse parts of the tree view; can select what to display in the Content pane from the tree view; and can select something in the `content view` which will add an open path to the tree view (perhaps dynamically changing the structure of the tree itself). The `ZTree` expands in 2D rather than in 1D, and zooming and shrinking interactions can be applied to make individual nodes larger and smaller. However, when the user causes a node to be opened, a degree-of-interest algorithm (DOI) tracks the user's attention and devotes more size to the most recent node. Thus manual sizing is possible but not necessary: desirable behaviour from the user's point of view as indicated in the Hyper Mochi study [27].

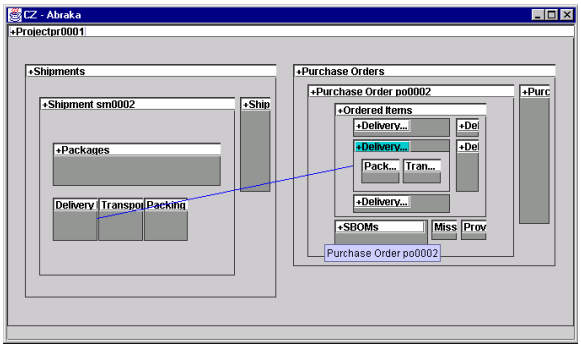
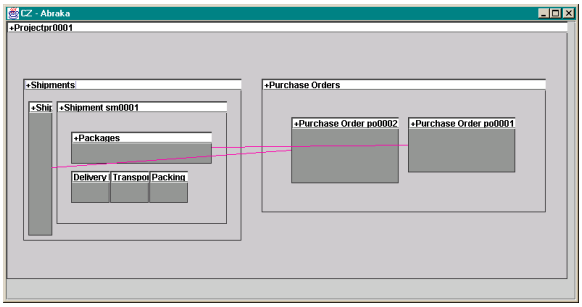


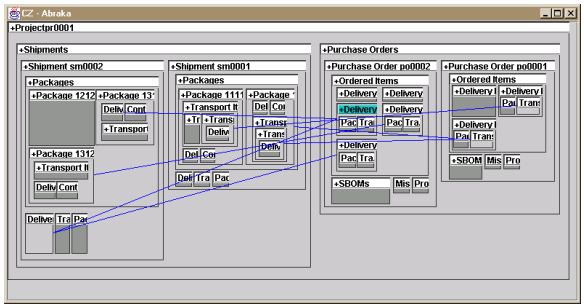
Figure 2. Expanding the `ZTree` structure through detail selections.

Items can be selected in the `ZTree` to view in more detail in the Content Pane. The `ZTree` node whose contents are currently displayed in the `content view` is considered to be the current *focus node* and its title bar is highlighted. Items selected in the `content view` affect the `ZTree` view in different ways. If the item has already been rendered (i.e., it has been laid out and specified), then the `ZTree` is opened along the relevant path in the tree structure to that item. However, if the appropriate item has not been specified and rendered it will be dynamically added to the `ZTree` structure and the `ZTree` layout will change to reflect the new node. If the item's canonical path is different from the path by which it was selected, the canonical path will be opened as well and a link drawn between the two related elements. This behaviour causes an incrementally derived structure view to be accumulated over the user's session as she explores other parts of the information space. In the current example, when the user selects item **di0203** in the `content view`, the `ZTree` will open up the associated path for that item and indicate the relationship between the two with a link (Figure 2).

It is important to note that this behaviour is supported by the `ZTree` but must be programmatically invoked: that is, the `ZTree` has no concept of a "canonical path", but it does have methods to define and render links based on related path criteria. The resulting view both explicitly renders the relation and reduces the navigational overhead required to explore the related context (in this example, the Purchase Order). Occasionally a diagnostic message will indicate that there is not enough space to actually open the nodes: the user can then close some other nodes to free up space or can resize the `ZTree` window.



(a)



(b)

Figure 3. Virtual links in the `ZTree`.

Previous versions of the CZ supported only simple links, which could cross levels of the hierarchy but were always rendered in detail. The resulting web quickly grew too cluttered. In the ZTree, links are themselves hierarchical and selectable objects.When a cluster node is closed, any links to its children will also be "closed" and rendered as a single *virtual link*. Just as one can drill down into the space by successively opening cluster nodes, one can also explore relations in successively more detail by opening the virtual links. Figure 3a shows a a `ZTree` view with virtual links: opening the links results in the more detailed view in Figure 3b. In

the interface these links are rendered in magenta and blue respectively. Finally, links also have a DOI which influences how much space they can have. This ensures that important links never get "squashed" between adjacent nodes.

The ZTree and CZ libraries support saving and restoring views. The derived structure displayed in the graphical map represents a composited history of the user's information forays (a graphical map of their information retrieval and problem-solving strategies). Users can thus recall their contexts over sessions, and can in fact share the maps with other users to highlight aspects of the information.

Layout: The Grid algorithm

The CZ algorithm has two inputs: the initial layout of the space, or *normal geometry*, and a set of scale factors (one for each node). The normal geometry and the scale factors are combined to produce the *zoomed geometry*, which is then displayed. In previous applications the normal geometry has remained constant (i.e., the space has been fully defined at runtime.) However, in hypermap applications such as the ZTree and CzWeb [11], the information structure can change over time as the user builds up successive paths through the information space. One approach would be to model the entire space and only render subsets of it. Indeed, the original CZ approach required the layout of the 2D space to be defined in an external map file. This is generally undesirable, as it restricts flexibility and introduces computational overhead. Instead we recalculate the normal geometry at each reconfiguration using an automatic grid-based layout algorithm.

2D automatic layout is an open research area in graph visualization (see [8] for an review). Force-directed (spring) layouts are common. However, in the ZTree, nodes are not necessarily fully connected by edges, necessitating a lattice structure to be superimposed to use a spring approach. Moreover, force-directed layouts cannot avoid overlap in all cases and do not distribute nodes aligned well with the axes of a bounding box. Instead, the ZTree layout problem can be seen as a variant of the bin-packing problem where the sizes of the bins are not known until the children are laid out.

We separate logical layout (topology and location) from pixel (x,y) layout. The CZ normal space consists of hierarchically organized clusters of nodes. We break this problem down into sub-problems of automatically laying out each group of siblings within a larger parent. Our logical layout approach (the Grid algorithm) partitions space into a rectangular grid of cells in which each cell is one logical unit. This grid is initially as close to square as possible, as we have observed that most reasonable layouts are achieved in grids that are either $x \times x$ or $x \times (x+1)$ units.

Grids can contain other Grids and have a 2D array of cells, a capacity (how many cells can be occupied in total), a weight (where $weight = \sum childCapacities$) and an orientation, since a Grid may need to be rotated to fit in the parent's available space. Each grid is optionally associated with some external (domain) object.

Child Grids can be added to the parent at any time. Children are sorted by size and inserted into the parent in a modified first-fit algorithm. Adding a child Grid to a parent may cause a "refit" of the parent: if there is not enough space in the parent, it will resize itself to accommodate the child and so on recursively up the hierarchy of Grids. If there is enough space in the parent to incorporate a rotated child, the child Grid will be rotated, and so will its children recursively down the tree. Nodes can be deleted in a similar fashion. There is no restriction on the size or number of Grids in a tree. Although theoretically this algorithm is potentially very costly, in practice the "close-to-squareness" constraint and the hierarchical partitioning of the solution space render it tractable, producing automatic layout calculation of ZTree spaces with no apparent performance lag.

We apply this in the ZTree by giving each node in the tree a Grid, laying out the space logically by recursively adding the child Grids associated with the ZTree node's children to that Grid, and then mapping the logical layout to pixel space (to account for borders, gaps between nodes, and other rendering issues.) The resulting layouts seem extremely workable. Overlap is impossible. All the layouts in the screen images in this paper were generated by this approach.

One potential disadvantage of this approach relates to the sorting of child nodes on size, since it can result in changed relative locations as new nodes are added. This can violate the consistency principle of maps discussed earlier. We are investigating variants which maintain relative layouts wherever possible.

6 Discussion, Issues and Future Work

At first inspection the ZTree seems to address the issues we identified for this application.It supports navigational maps which include both hierarchical and relational information (Figure 5). It allows users to drill down in the information space without sacrificing context using both node-centric (entity) and link-centric (relation) access. View traversal and navigation are aided both by persistent context and an automatic layout which preserves relative positioning necessary for effective perceptual processing. User intervention in resizing views or in re-arranging the automatic layout is supported but unnecessary. Composite graphs are built up of user information retrieval allowing the user to recall higher-level problem solving contexts and to share them with other users.

However, this is very preliminary work, and many questions remain. The obvious one is: Do the users like

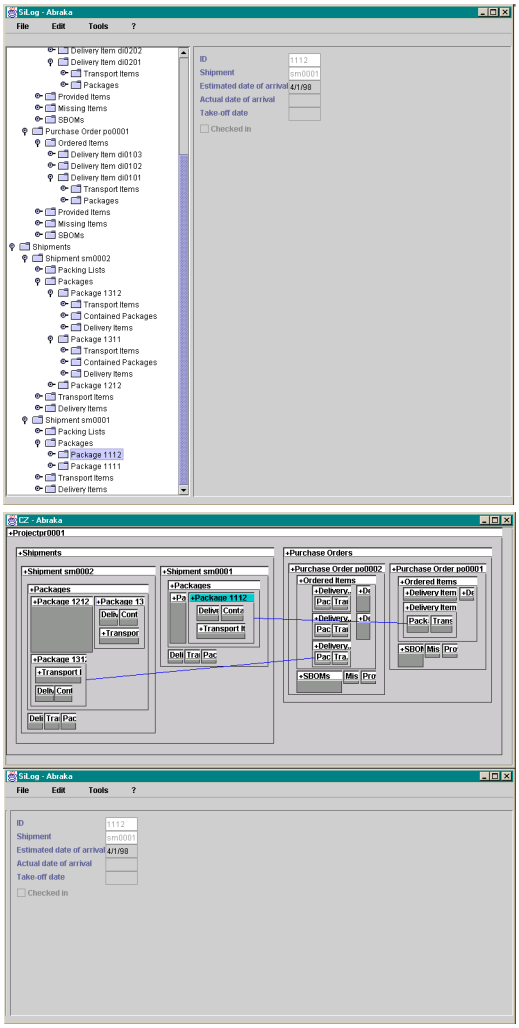


Figure 4. JTree and ZTree maps for the same paths.

it? Is it useful? We have added the ZTree as an alternative user interface technique to the existing SiLog application based on initial user feedback. We are hoping to arrange more detailed studies with field-based IT managers in the future. In the interim, we raise some open questions about the extent of ZTree utility.

Node representation and DOI: Each object (node or link) has an interest measure associated with it which is altered by user attention or by application specific factors. The DOI affects an object's chances of getting screen space. In previous CZ applications that has been useful since we use the node itself to contain information. However, as a structure view, it may make more sense to tune the DOI to other measures such as how many elements it contains, or how important it is with respect to the application, or how many times the user has visited it. Moreover, it introduces the potential for using the object space to convey information about the detailed view. How can we exploit this in a database application, and would it be useful?

Space requirements: As Figure 5 shows, the ZTree can accommodate more contextual information than a scrolling list. Eventually, however, it requires more screen space than is available.Thus previously accessed nodes and links may have to be pruned from the view. There are issues to be decided in how we go about this. Do we automatically prune the view based on factors like age (least recently accessed), DOI, or distance from current focus point? Do we prompt the user to free up space and let her make the decision? This will have to be tested.

Representing links: While we feel that link manipulation and representation has great potential for facilitating database navigation and comprehension, there is as yet little knowledge and experience on how best to approach this. We hypothesize that in many complex information worlds, relations between entities are more interesting than the entities themselves. We anticipate much interesting research in this area.

7 Acknowledgments

This work was supported by a research grant from ABB Corporate Research, Heidelberg. We are indebted to our colleagues Anne Tissen at ABB and Dr. Thomas Strothotte at the Technical University of Magdeburg for their interest and feedback.

References

[1] Andalman, B.A., Ryall, K., Ruml, W., Marks, J. and Shieber, A. "Design Gallery Browsers Based on 2D and 3D Graph Drawing", in *Proceedings of the 5th International Symposium on Graph Drawing*, ed. DiBattista,G. , Springer, 1997, pp. 322-329.

[2] Asahi, T., Turo, D., Shneiderman, B. "Using Treemaps to Visualize the Analytic Hierarchy Process", Information Systems 6(4), Dec. 1995, pp. 357-375.

[3] Bartram, L., Ho, A., Dill, J. and Henigman, F. "The Continuous Zoom: A Constrained Fisheye Technique for Viewing and Navigating Large Information Spaces", in *Proceedings of UIST '95*,ACM, NY 1995, pp. 207-214.

[4] Bartram, L., R. Ovans, J. Dill, J. Dyck, A. Ho and W. Havens. "Contextual Assistance in User Interfaces to Complex, Time-Critical Systems: The Intelligent Zoom." *Proceedings of Graphics Interface '94*, pp. 216-224, May 1994.

[5] Bederson, B.B. and J.D. Hollan. "Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics," *Proceedings of UIST '94*, November 1994.

[6] Bederson, B. B., Hollan, J.D., Stewart, J., Rogers, D., Druin, A., and Vick, D. “A Zooming Web Browser”, SPIE Multimedia Computing and Networking **2667**, 1996, pp.260- 271.

[7] Carpendale, M.S., Cowperthwaite, D and Fracchia,F.D. “Editing in Elastic Presentation Spaces”. Submitted to UIST ‘99.

[8] di Battista, G., Eades, P., Tamassia, R. and Tollis, I. Algorithms for Drawing Graphs: An annotated bibliography.” Computational Geometry Theory and Applications, 4(5):235-282, 1994.

[9] Eades, P., Cohen, R.F., and Huang, M.L. “Online Animated Graph Drawing for Web Navigation”, in Proceedings of the 5th International Symposium on Graph Drawing, Springer, 1997, pp. 330-335.

[10] Fisher, B., Agelidis, M,. Dill, J., Tan, P., Collaud, G., and Jones,C. “CZWeb: Fish-Eye Views for Visualizing the World-Wide Web”, in Proceedings of HCI International ‘97.

[11] Fisher, B. and Dill, J. “Application of theories of indexical cognition to a Web-based workspace.” *American Association for Artificial Intelligence Symposium on Smart Graphics*, May 2000.

[12] Furnas, G.W. “Generalized Fisheye Views.” *Proceedings of ACM SIGCHI’86*, pp. 16-12, April 1986.

[13] Furnas, G. “Effective View Navigation”, in *Proceedings of CHI ‘97 Human Factors in Computing Systems*, ACM/SIGCH, N.Y., 1997.

[14] Hemmje, M., Kunkel, C., and Willet, A. “LyberWorld - A Visualization User Interface Supporting Fulltext Retrieval”. in *Proceedings of ACM SIGIR 94*, ACM Press, New York, 1994.

[15] Hollands, J.G., T.T. Carey, M.L. Matthews and C.A. McCann. “Presenting a Graphical Network: A Comparison of Performance Using Fisheye and Scrolling Views.” In *Designing and Using Human-Computer Interfaces and Knowledge-Based Systems*,. G. Salvendy and M. Smith (Eds), Elsevier, pp. 313-320, 1989.

[16] Jul, S. and Furnas, G. “Navigation in Electronic Worlds”, *Report on the CHI 97 Information Navigation Workshop*I. ACM/SIGCHI 1997.

[17] Lai, Y. and Waugh, M. “The Effects of Three Different Hypertextual Menu Designs on Various Information Searching Activities”. *Journal of Educational Multimedia and Hypermedia*, **4(1)**, March 1995.

[18] Laurini, R. and Thompson, D. Fundamentals of Spatial Information Systems. Academic Press, London 1992.

[19] Mukherjee, S., Foley, J. and Hudson, S. “Visualizing Complex Hypermedia Networks Through Multiple Hierarchical Views.” *Proceedings of CHI ‘95*, pp. 331-339, 1995.

[20] Rapley, M. H.and Kennedy, J. B. “Three Dimensional Interface for an Object Oriented Database” in I*nterfaces to Database Systems. Lancaster 1994*, Sawyer, P., Ed.; Springer: 1995, 143.

[21] Robertson, G.G., Mackinlay, J.and Car, S.K. “Cone Trees: Animated 3D visualizations of hierarchical information.” *Proceedings of CHI ‘91 Human Factors in Computing Systems*, ACM/ SIGCHI, 1991, pp. 189-194.

[22] Sarkar, M., S.S. Snibbe, O.J. Tversky and S.P. Reiss. “Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens.” *Proceedings of ACM UIST*, pp. 81-92, Nov. 1993.

[23] Schaffer, D., Z. Zuo, Greenberg, S., Bartram, L., Dill, J. Dubs, S and Roseman, M. “Navigating hierarchically clustered networks through fisheye and full-zoom methods. ACM Transactions on Computer-Human Interaction, 3(2):, 1996, pp. 162-188.

[24] Tweedie, L. “Characterizing Interactive Externalizations”, in *Proceedings of CHI ‘97 Human Factors in Computing Systems*, ACM/SIGCH, N.Y., 1997, pp. 375-381.

[25] Wills, G.J. “NicheWorks - Interactive Visualization of Very Large Graphs”, in Proceedings of the 5th International Symposium on Graph Drawing, Springer, 1997, pp. 404-414.

[26] Nielsen, J. “The Art of Navigating Through HyperText”. *Communications of the ACM*, **33(3)**, March 1990.

[27] Toyoda, M. and Shibayama,E. “Hyper Mochi Sheet: A Predictive Focusing Interface for Editing and Navigating Nested Networks Through a Multi-Focus Distortion-Oriented View”. *Proceedings of CHI ‘99 Human Factors in Computing Systems*, ACM/SIGCHI, 1999, pp. 504-511.