Snap-Together Visualization: Coordinating Multiple Views to Explore Information

*Chris North and Ben Shneiderman**
Human-Computer Interaction Lab, Institute for Advanced Computer Studies,
*Institute for Systems Research, Department of Computer Science
University of Maryland, College Park, MD 20742 USA
north@cs.umd.edu, ben@cs.umd.edu
http://www.cs.umd.edu/hcil

ABSTRACT

Information visualizations with *multiple coordinated views* enable users to rapidly explore complex data and discover relationships. However, it is usually difficult for users to find or create the coordinated visualizations they need. Snap-Together Visualization allows users to coordinate multiple views that are customized to their needs. Users query their relational database and load results into desired visualizations. Then they specify coordinations between visualizations for selecting, navigating, or re-querying. Developers can make independent visualization tools 'snap-able' by including a few hooks.

KEYWORDS: User Interface, Coordination, Multiple Views, Tightly Coupled, Information Visualization.

INTRODUCTION

The *multiple coordinated views* approach is a powerful and increasingly-employed user-interface technique for exploring information. Each view is a visualization of some part of the information, and views are coordinated (a.k.a. "tightly coupled", or "linked") so that they operate together as a unified interface [NS97].

For example, Microsoft Word's document-map feature displays the table of contents of the text in an adjacent frame. Selecting a heading in the map scrolls the document text directly to that section. Likewise, scrolling the document text highlights the current section in the map. A second example, Windows Explorer actually has 3 views: The left pane contains the directory hierarchy, the right pane shows the detailed contents of a selected directory, and (with "View as Web Page" on) also displays details of a selected file including a miniature quick-view. In a more advanced visualization, selecting a record in Spotfire's [AS94] starfield displays its attributes in a web browser or the integrated details pane. On the web, frames are used like Word's document map to link tables-of-contents to main views. However, frame coordinations are only one-directional due to their use of the hypertext model.

Multiple coordinated views have many advantages that have been documented in user studies [NWS86] [CWM94] [SSS86]. Coordination improves user performance. For example, users can select items in overviews (such as a table of contents) to produce details (such as the full chapter) in an adjacent window or to synchronize scrolling in a French translation. Coordination users can also exploit relationships to facilitate exploration across information types. For example, clicking on a city in a map yields a city description, a calendar of events, and photos in adjacent views.

However, users often cannot coordinate the visualizations they need. The choice of visualizations and coordinations are highly dependent on the information and tasks. Some common coordinations have been implemented, but others require custom programming. Development tools, which focus on the independent view approach, do not support building coordinated views easily. Hence, a mechanism is needed to give users the coordinated multiple-view interfaces they want, yet, at the same time, save designers from endless development of coordinations.

SNAP-TOGETHER VISUALIZATION

Snap-Together Visualization (STV) is an architecture and system that allows users to coordinate visualization tools. Users can build coordinated multiple-view interfaces for information exploration that are customized to the specific needs of their data and tasks. Users query their data and load results into desired visualizations, then establish coordinations between visualizations for selecting, navigating, or re-querying. Developers can make independent visualization tools 'snap-able' by simply including a few hooks with a small amount of code.

Our goals for designing STV are:

1. Coordination: Allow users to coordinate views to build their own multiple-view user interfaces for exploring information.
2. Utilize third party visualizations: Allow users to take advantage of visualization tools built by themselves or other researchers and developers.

Model

To support the first goal, the coordination model in the STV user-interface is based on the unit of information, called an *object*. There are four major concepts in this model:

Information is stored in an underlying relational database. An object represents a tuple. Since users explore information by following relationships (e.g., select a directory to discover its contents), the relational model provides a good basis for coordination. It also provides a robust general data format and a unique identifier for each object. An objectID is the tuple's primary-key value.

A *visualization* is a view (graphical and/or textual) of a set of objects (e.g., a scatterplot of data points or text view of the sections of a document). A query (e.g., SQL) extracts the records from the database and loads them into the visualization.

User *actions* act on individual objects or sets of objects in a visualization. There are three major categories of actions:

1. *Select* an object. E.g., click on, mouse over, rubber band, etc. Visualizations typical respond by visually highlighting the selected object(s).
2. *Navigate* to an object. E.g., scroll to, zoom onto, open subtree of, etc.
3. *Query* (based on an object). Query and load a new set of objects into the visualization (by setting the query's parameters based on a given object). This allows navigation through other information in the database. E.g., load the list of files contained in a given folder.

While exploring, users indicate objects of interest by selecting or navigating to them. The system responds to reveal related information by selecting, navigating, or querying. (We focus on information exploration, not manipulation or editing tasks.)

Coordination maps actions on objects in one visualization to actions on objects in another. The two visualizations are coordinated so that when one of the actions occurs, the mapped action is also executed in the other view. E.g., selecting a section title in the table-of-contents outliner scrolls the document text to that section.

User Interface

When users open a database with STV, the menu window (Figure 1) displays a list of all tables and queries stored in the database, and a list of available visualization tools. Queries can be added or edited using the database's visual query editor or typed in as SQL. Users can then build coordinated multiple-view interfaces by simply opening appropriate visualizations and 'snapping' them together:

1. Open visualizations via queries. When users select a table or query from the menu and drag and drop it onto a visualization in the menu, the visualization opens in a new window and the results of the query are loaded in. STV adds a menu to the visualization window for various STV-related actions. To put a different query result into the window, users simply drag a new query from the menu window to this menu. The visualization displays the data and users can interact with it as a stand-alone application.

2. Snap visualizations together. Users can then

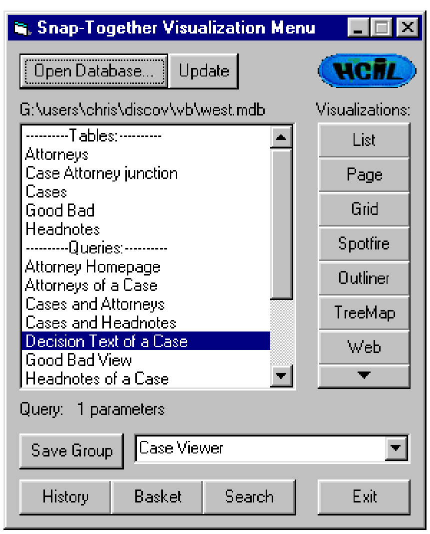

Figure 1: The Snap-Together Visualization window lists tables and queries in the database and displays a menu of available visualizations.

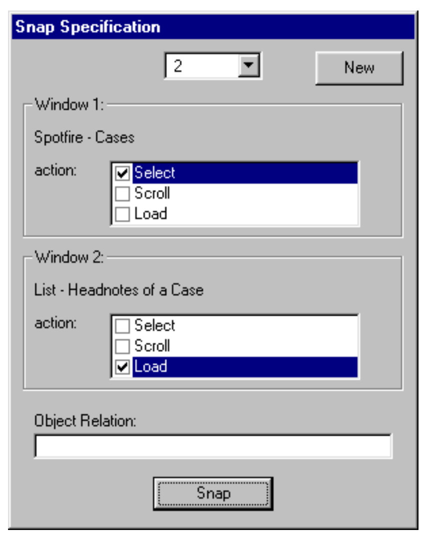

Figure 2: In the Snap Specification dialog, users select which actions to coordinate between two views. In this example, selecting a case in Spotfire will load the headnotes of the case in a textual list view.

coordinate any 2 visualizations by a drag-and-drop action between the STV menus of both visualization windows. This opens the Snap Specification dialog box (Figure 2), and users select which actions in each window to coordinate. Then, the 2 visualizations are tightly coupled such that interacting in one causes the desired effects in the other. The coordination between the visualizations can be modified or deleted using the STV menu on either window.

Any number of visualizations can be opened and coordinated in this way, allowing users to construct exploration interfaces consisting of many integrated views. They can then browse their information, and all views maintain synchronization. At any time, if an additional view is needed, users can simply snap it in.

STV creates a new class of users called "coordination designers". Coordination designers know the data, have knowledge of good user interface design, and use STV to construct coordinated multiple-view interfaces. Coordination designers can be end-users who snap visualizations together for their own browsing needs. Or, coordination designers, like web designers, can build browsing interfaces for other less-knowledgeable end-users to explore information. Coordinated designs can be saved, distributed, and then broadly used by others.

Scenario

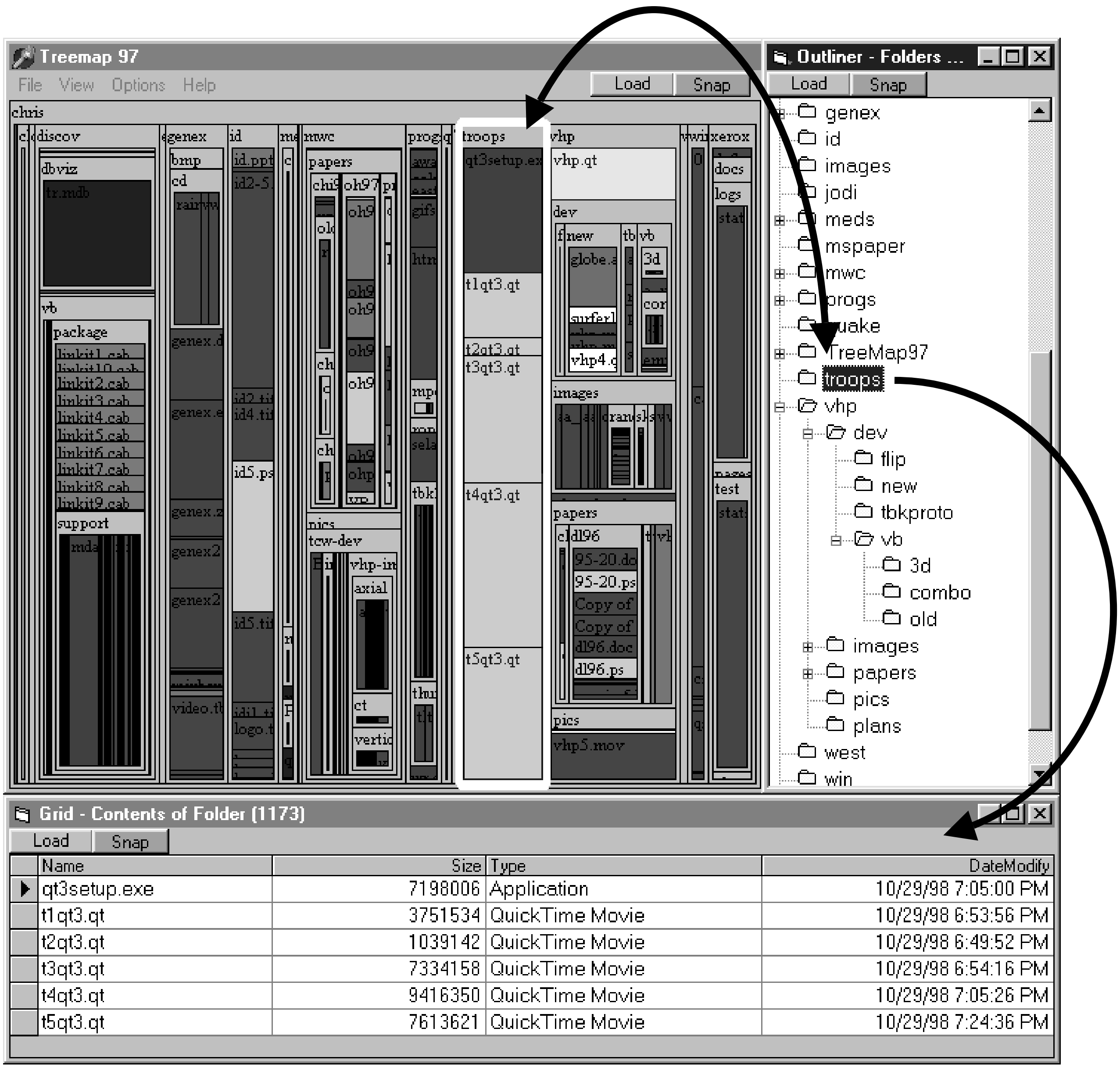

Figure 3: A coordinated multiple-view interface, created with Snap-Together Visualization, for exploring directory structures. Selecting a directory in the treemap (left) also selects it in the outliner (right) and displays its contents in the grid (bottom).

While Windows Explorer is adequate for some users, system administrators may benefit if a treemap [Shn92] visualization were included in its set of views. Treemaps, which show hierarchies in a recursive slice-and-dice pattern, are strong in revealing overall layouts of entire large hierarchies. Administrators can quickly gain an overview of directory structures to spot buried large or old files and discover duplicated similar directories (Figure 3).

From a database of a file system, drop the table of directories into an outliner visualization to display the directory hierarchy. Drop a query of files for a given directory into a spreadsheet grid view, and then snap the two views together, coordinating between the select action in the outliner to query in the grid. Then, selecting a directory in the outliner displays its contents in the grid. To take advantage of more advanced visualization tools, simply snap in a treemap view by dropping the files and directories table onto the treemap and then coordinating the select action to the outliner. Then, selecting a directory in the treemap highlights it in the outliner, which then displays its details in the grid. Now administrators can use the treemap to quickly find suspiciously large groups of files, easily see where they are in the file structure in the outliner, and immediately access details of files in the grid to discover which user is hogging space.

Administrators can easily extend their exploration environment as needed. For example, to identify directories to archive, they might snap in a Perspective Wall [MRC91] visualization for a timeline view. Snap a Spotfire scatter plot, mapping file creation date to the X-axis and last access date to Y-access, to locate unused files on the diagonal. To examine contents of many files, snap a file viewer to the grid view. Then, they can quickly peak into files by simply selecting them in the grid. At the opposite end of the scale, administrators might include an overview of their network, allowing them to select any particular file structure to browse in the rest of the views already created.

Other Features

To demonstrate other features of the STV system, we use a richer, more complex scenario. STV has immediately proven its value in our information visualization projects. For example, WestGroup provides information support for legal professionals, including databases of millions of court cases from Federal and state jurisdictions. We used STV to quickly prototype multi-view visualizations to help

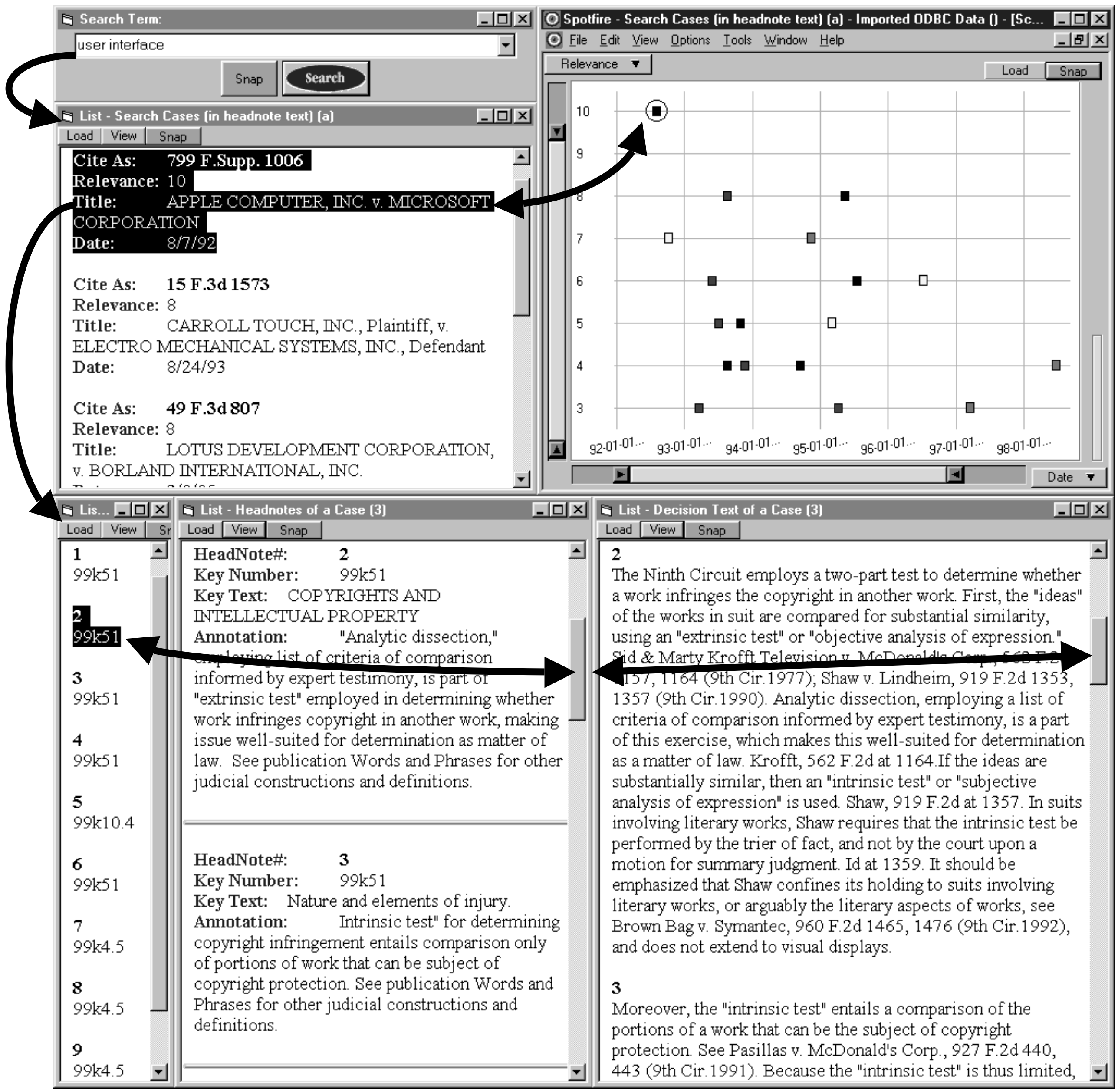

Figure 4: Exploring legal information with a searcher's workbench created using Snap-Together Visualization. The Search Term box executes a search query and loads the results into the text view and Spotfire. Selecting a case in Spotfire highlights the title in the text view and loads details of the case into the Case Viewer (bottom). The Case Viewer is composed of 3 views (left to right): case overview, headnotes, and decision text. Scrolling in the headnotes and decision text is synchronized, and selecting in the overview scrolls both.

WestGroup explore user interface alternatives for different "workbenches" for different types of users.

We first demonstrate building a Case Viewer (Figure 4, bottom). A case is composed of a judge's decision text, which is partitioned into sections. Each section has a headnote, consisting of a categorization (in a taxonomy of case law) and annotation. The existing user interface simply lists out all the information in a single web page with many intra-links between sections and headnotes. Since users often refer to headnotes while browsing the decision text, yet need to scan the decision as a contiguous text, a two-view synchronized-scrolling approach is appropriate. The headnotes and decision text are queried into separate adjacent textual list views. The scroll actions of both views are coordinated. Then, as users scroll in either window, the other always shows the related information.

Since many decision texts are long, containing 10 to 50 headnotes, a good HCI designer might include a small overview frame for quick access to any section. Query only the section numbers into another list view, and coordinate its select action to the scroll-to action of the Headnotes view. All three windows become coordinated. Scrolling either detail view highlights the current section number in the overview, and vice versa.

To make the Case Viewer into a general tool, parameterize the queries of each view to load a case given its caseID, using an SQL component like "where decisions.caseID = ?". Then, create query-to-query coordinations between them so that when a case is loaded into one, it is loaded into all 3.

Save Groups: From the STV menu, the coordinated set of windows can be saved as a group and given a name such as "Case Viewer". Then, at any time, selecting the group name from the STV menu will launch a new instantiation of the group of 3 coordinated windows. This gives users the ability to construct new composite visualization tools and reuse them as snap-able primitives.

Search Box: Users typically browse cases by search terms. To search for cases, a search query that takes a search phrase as a parameter and returns a list of hits sorted by relevance is dropped into a textual list view to show results. STV provides a simple search box window that can be snapped to such a results window by coordinating the search action to the result view's query action (Figure 4, top). Entering a search term such as "user interface" in the search box reveals relevant case titles in the results view.

Then, users can quickly browse the case hits by simply snapping on a Case Viewer and coordinating the result view's select action to the previously-constructed Case-Viewer's query action. If search results are large, users can snap in a visualization tool, such as Spotfire, to help them spot trends. Selecting a recent and highly relevant case in Spotfire's scatterplot highlights the corresponding title, "Apple v Microsoft", in the textual results view and displays the details in the Case Viewer (Figure 4).

Shopping Basket: When exploring such a large database of cases, users can gather a set of interesting cases into an STV shopping basket window by drag and drop.

History: In designing user interfaces for information exploration, we increasingly recognize the need for history keeping, allowing users to review previous states. An interesting bonus of the STV architecture is that, since it receives notification of user actions in all visualizations, it can easily keep track of the history. The STV history window displays the history list of actions. Selecting an action from the list replays it.

Extract: Of course, the reason users must explore information is to extract the knowledge required to accomplish some task. For example, an attorney might need to contact other attorneys of recent similar cases. In a Spotfire display of attorneys of similar cases, the attorney selects 25 recent attorneys from the scatterplot, and drag-and-drops the them onto the "To:" field of an email message window. STV then presents a small popup list of the available fields in the dropped records (Figure 5). The attorney selects the "email address" field, and STV drops all the email addresses of the selected attorneys into the field. The attorney then types a message and sends it off. STV can drop or paste to any OLE compliant window and extract any fields from records. For visualizations that do not provide drag-and-drop capability, STV initiates the drag itself and provides the data since it tracks selection actions.

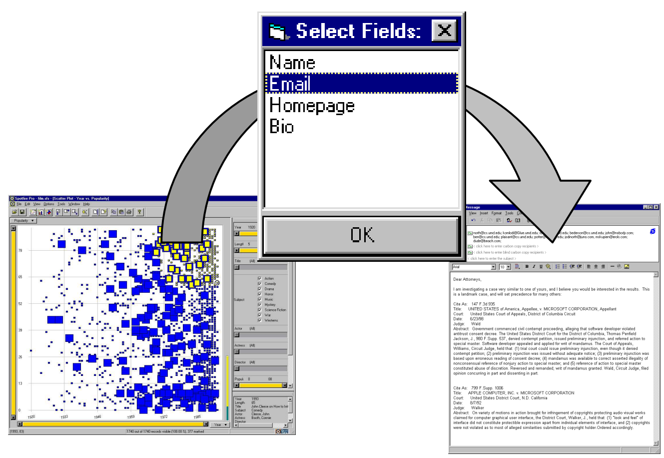

Figure 5: Users can extract data to a composition window, such as an email editor, by drag and drop and then select fields to include in the pasted data. For example, users could drag attorney records from a scatterplot to an email message and select to extract only their email addresses.

ARCHITECTURE

STV is a centralized software system that acts as an intermediary among the visualization tools and the database (Figure 6). The visualization tools are actually independent software programs, potentially built by 3rd party vendors, augmented only slightly to integrate into the STV environment via inter-process communication. The database is also an independent entity generated by standard relational database software. To edit queries, we employ the software's SQL or visual query editor.

When users open a query result into a visualization, several operations are executed. First, STV executes the query and receives a recordset, and the selected visualization tool is launched. Then, if needed, the recordset is translated into the input format required by that visualization tool, stored to a temporary file, and the "Load" method of the visualization is invoked. The visualization behaves as it normally would, reading in the data file and displaying it. Database-enabled visualization tools can bypass the query execution, translation, and storage steps by accepting a connection and query string directly. STV adds a small menu palette to the visualization's window for STV-related actions by inserting a small child window. This reflects our vision of STV as integrated into the window- or frame-management system. Hence, user actions in this palette invoke STV software.

A visualization tool has a set of user actions that are snap-enabled. Typically, these are select and navigate actions. At initialization, the tool communicates this set to STV. When a pair of visualizations are snapped together, a mapping is defined as:

$$(viz_A, action_A, objectID_A) \Leftrightarrow (viz_B, action_B, objectID_B).$$

The pair (viz_A, viz_B) are the visualizations being snapped together, determined by the source and destination visualizations of the user's drag-and-drop snap action. The pair ($action_A$, $action_B$) is specified by the user in the resulting Snap Specification dialog box. In most cases, $objectID_A = objectID_B$, as in primary-key to foreign-key joins. However, to allow for more complex coordinations as in data mining applications, they can be related by an arbitrary relational transformation.

During execution, visualization tools communicate with STV:

1. Action notification. When users perform an action in the visualization, a notification message is sent to STV containing the triple: (visualizationID, action, objectID).
2. Action invocation. STV can send a message to a visualization to programmatically invoke an action normally invoked by the user, as in visualizationID.execute(action, objectID).

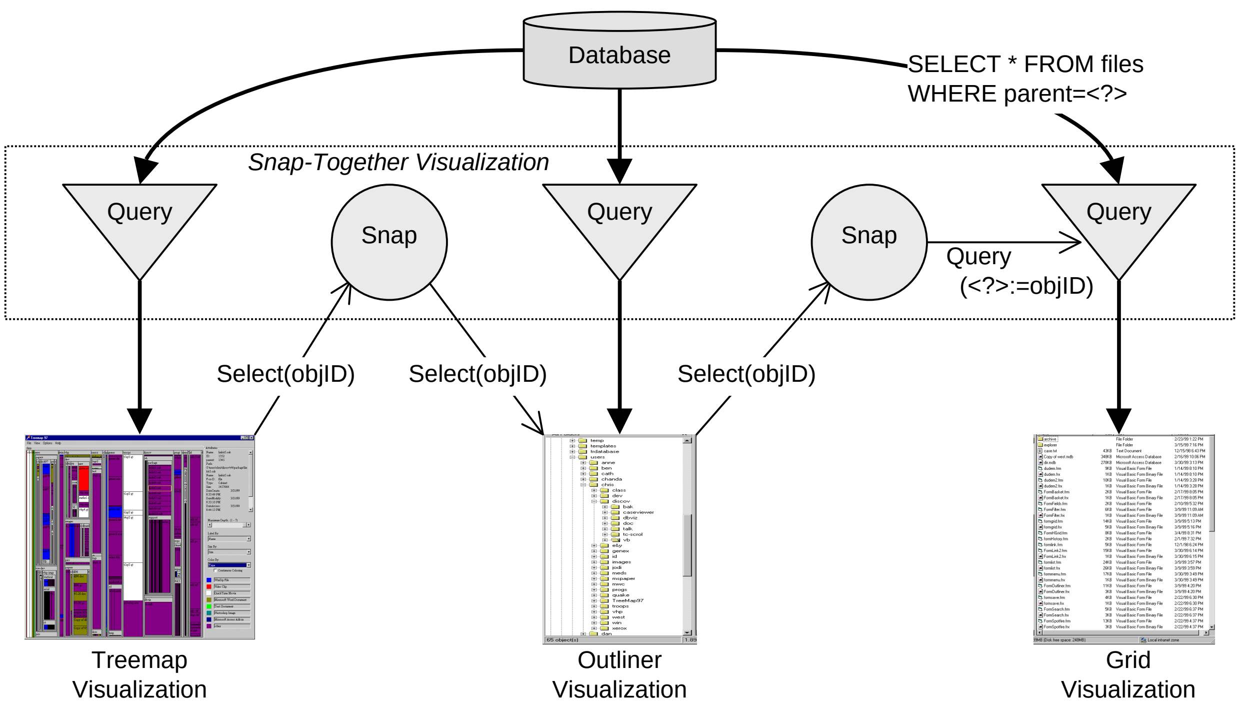

Figure 6: The Snap-Together Visualization architecture. Users select queries to load data into visualizations. Then, they snap visualizations together to coordinate actions between them. This example demonstrates how the actions propagate in the interface in Figure 3 when users select a node in the treemap. The same node is selected in the outliner, and then the query to the grid view extracts the children of that node from the database and loads them into the grid.

When users perform $action_A$ on $objectID_A$ in visualization viz_A, notification is sent to STV. In turn, STV applies the transformation and invokes $action_B$ on $objectID_B$ in viz_B. Select and navigate actions are sent to the visualization for execution. Query actions are sent to the visualization's input query, causing the query to be re-executed using $objectID_B$ as query parameter value, and then invoke the visualization's load operation.

As users snap pairs of visualizations together, STV maintains an internal graph data-structure representing the coordination graph. Nodes in the graph are visualizations and links are the snap mappings between them. When notified of a user action in a visualization, STV traverses the graph, invoking the coordinated actions on the linked visualizations.

To illustrate with the treemap+outliner+grid example (Figure 6), when users select node X in treemap, it sends a (select, X) message to STV. STV traverses the coordinations set up earlier by the user, and sends the mapped message to the outliner (select, X). Recursing, STV then sends (query, X) to the Query input for grid. The grid's SQL query is: "SELECT * FROM files WHERE files.parent = ?". The parameter "?" binds to X, the query is executed, and the resulting list of files in folder X are loaded into the grid.

Coordination Properties

Coordinations are:

- *Commutative* (bi-directional). Given a snapped pair of visualizations, user action in either visualization causes action in the other.
- *Transitive*. If visualization A is snapped to B and B is snapped to C, then a user action in A will produce action in B, which will then produce action in C.

Common coordinated view pairs correspond to data relationships and other properties of queries. We illustrate with examples from the WestGroup scenario.

- *Synchronized scrolling*: Navigate-to-navigate actions on a one-to-one relationship with a common sort order. E.g., Headnotes and decision text (Figure 4, bottom right and center).
- *Overview and detail view*: Select-to-navigate on one-to-one relationships where the overview query selects fewer or smaller fields. E.g., Headnote numbers (one small field) to headnote text (includes lengthy annotations). (Figure 4, bottom left and center)
- *Hierarchical browsing*: Select-to-query on one-to-many relationships to browse across levels of scale. Different visualizations may be appropriate at different levels. E.g., Spotfire display of cases to textual list of headnotes of a selected case (Figure 4, top right and bottom center).
- *Brushing*: Simple brushing is select-to-select on one-to-one relationships. E.g., Spotfire view of cases to the textual list of the same cases (Figure 4, top right and left). More interesting is brushing across a many-to-many relationship. E.g., Selecting cases in one view to highlight attorneys involved with those cases in another view.

Making Visualizations 'Snap-able'

The second goal of Snap-Together Visualization is to allow users to take advantage of the wide variety of helpful visualization tools that have been implemented by researchers and developers. A closed system that only uses visualizations constructed internal to the system would be merely a small improvement over a hard-coded multi-view user interface, would quickly lead to obsolescence, and therefore would not solve the problem. Hence, we minimize effort required to make any off-the-shelf third-party visualization 'snap-able' to the extent that even a fourth party could easily accomplish.

STV is analogous to the standardized cut-and-paste or drag-and-drop feature of modern windowing systems. A centralized server, integrated into the window system, handles much of the work. Then, with the addition of a few simple hooks and a small amount of code, a powerful feature is enabled in an application, making it interoperable with many others. Effort is low and payoff is large.

To make STV open, and minimize impact on visualization tools, we make several assumptions about the tools. They:

- Remain independent software entities, need not be compiled into STV, run as independent processes, and can be run as stand-alone applications.
- Use only simple inter-process communication with STV to send and receive events as (action, objectID) pairs.
- Must support only the actions they already support (e.g. select). No major new functionality required.
- Communicate only in terms of object IDs. They identify and act on an object by its unique ID, and cannot query or search on attributes.
- Are not aware of the larger data context of the external STV database, only of the data currently loaded into them by STV.
- Accept input data in their own format.

To enable a visualization, four hooks are required:

1. Initialization. At initialization, notify STV of available actions for coordination, such as select or scroll.
2. Action notification. When a user action (e.g., mouse click to select an object) is processed, send an event to STV, passing the objectID of the selected object.
3. Action invocation. A method, externally invokable by STV, programmatically executes a given action on a given objectID (e.g., highlight a selected object). This may require some code to search the internal data structures for objectID.
4. Load. A method, externally invokable by STV, initiates the existing routine to load data into the visualization, potentially from the given temporary file.

Data structures may need to be augmented to handle objectID's.

To snap enable the treemap visualization tool, which was originally developed by others, required approximately 2 hours of work for us to add approximately 20 lines of code to its software. Some well-designed component-based visualizations, such as Spotfire, already support a full suite of methods and events. In this case, access to the source code is not necessary. STV provides a template for a simple wrapper program that translates the STV communication protocol to calls to the visualization component.

The only other requirement is a translator program that converts the input data from the record set format to the input format of the visualization tool. However, we claim this as a gain, not a cost, because only one such translator ever has to be written for each visualization tool. From the users' point of view, this is a big advantage because traditionally users must write their own translators for each visualization they use. With STV they need at most one: to convert their data into a relational database. And visualization developers need to supply only one: to convert a record set to their visualization's format.

IMPLEMENTATION

We have implemented the STV architecture as described in this paper. It is currently developed on the Windows platform, using ODBC for database access and COM for inter-process communication. Any database software could be used to generate the databases. We use Microsoft Access. To edit queries, Access's visual query design tool is more than adequate.

STV is motivated by our current work with WestGroup and the US Census Bureau, and other previous projects. It was well received by professionals in these communities, and WestGroup is integrating STV concepts into their systems.

FUTURE WORK

The architecture could be extended in several ways. Multiple objects per action are needed to handle multiple selections. In addition, generalized multi-way coordinations with multiple-parameter queries would allow for several views to participate in a single coordination, such as simultaneous menus. For example, users could select a county in one view, an industry from a second view, and a year from a third view, in any order, to view production statistics in a fourth view. Multi-object coordinations could also be accomplished by including attribute-based coordination, e.g., select objects where 10 < object.attr < 25. Some visualization tools, such as dynamic query tools, may be able to support such expressions. This could also be useful for visualizations that do aggregation (e.g., histograms).

In terms of the user interface, three major improvements are needed. First, better window management is needed for tiling, docking, or grouping windows together [KS97]. For example, a Case Viewer might be grouped as one window composed of three sub-frames. Second, a visual coordination-editor tool might combine the querying and coordinating steps into a more task-oriented process. Given one view, what else can I explore from here? Third, a visual map is needed to indicate what coordinations are in place between a set of snapped views.

LIMITATIONS

Clearly, STV places a premium on screen space. This research anticipates large desk-sized displays. We often use a two-monitor workstation to demonstrate coordinating many views for information-rich interfaces.

The STV coordination model is based on user actions on information objects. It is not well suited for coordinating visual layout and image browsing [PCS95]. Examples include Adobe PhotoShop's overview window and the InfoMural [JS95], where views paint themselves based on pixel-level information in other views.

The architecture reveals a tradeoff between scalability and the use of independent visualization tools. There is a bottleneck in the load operation of some visualizations. The *slowest common denominator* problem can arise if interaction is limited to the speed of the slowest visualization.

RELATED WORK

Several other systems include some capability for users to coordinate multiple views. Each is hardwired for one or two specific coordinations, but allows users to configure options in the coordination or choose windows to participate. All of them use a fully internal architecture in which visualizations must be built within the system using its internal shared data storage and functionality. The only exception is Cyberdesk [DAP97], which allows users to select text in any window, and then choose a "service", such as a web search or address book, from a menu to send the text to the service's window.

Most common are systems for brushing scatterplots [BC87], in which painting data points in one plot also paints them in the others (e.g. select to select). XGobi [BCS96] provides significant options for brushing, such as accumulation, color, glyphs, etc. XmdvTool [WA95] stands out in its ability to brush points (object-based) or regions (attribute-based). With Visage's [RLS96] "information-centric" approach users can drag-and-drop objects between views and brush them. Its SAGE component overcomes the problem of a limited set of visualizations by generating custom data visualizations automatically.

With LinkWinds [JBO94], users connect controls and views to build a series of filters for scientific visualization.

DEVise [LRB97] coordinates region selections and axes of graphs to synchronize zoom and pan. In the Apple Dylan programming environment [DP95], users browse hierarchical objects by splitting and linking frames so that selecting a folder in the source frame displays its contents in the destination frame. Logos, a commercial bible software package, can coordinate sets of views of different translations and commentaries to synchronize scrolling by verses. Spreadsheet Visualization [CBR97], a unique approach, arranges views as cells in a grid. Then, users can apply algebraic operations between rows of visualizations.

Of these systems, LinkWinds and Dylan use a drag-and-drop action to select windows for coordination. Others use selection from window lists. LinkWinds is the only to visually indicate coordinated windows, by drawing lines between them.

Some inspiration for the STV model comes from RMM [ISB95], a system for constructing web sites from underlying relational databases. In RMM, relationships identify hypertext navigation structures, whereas in STV, relationships correspond to coordinations.

CONCLUSION

Snap-Together Visualization has many benefits for both visualization developers and users.

For visualization developers:

- Reuses visualizations. STV treats individual visualization tools as components. Each visualization needs to be developed only once.
- Simplifies visualization development. Developers can focus efforts on the primary view of the visualization and do not need to incorporate supporting views. STV can supply those.
- Eliminates the need to develop multiple-view coordinations.
- Steers developers to more rigorous identification of the purpose and strengths of each visualization tool. E.g. for what situations should users snap in visualization X?

For users and coordination designers:

- Offers advantages of coordinated multiple-view interfaces, including improved user performance and rapid exploration across information types.
- Coordinates views without programming.
- Customizes exploration environment to specific data and tasks.
- Accesses many visualization tools, and multiple-view composites shared by others.
- Uses single data input format.
- Enables rapid prototyping of workbenches.
- Provides an appropriate user interface when distributing data to end-users.

Information visualization researchers and developers have built many helpful visualization tools. With the benefits listed above, STV users can utilize these visualizations to build their own coordinated multiple-view user interfaces for exploring information.

ACKNOWLEDGMENTS

This research is supported in part by funding from WestGroup and the US Census Bureau. Thanks to Jerome Brown and Shaun Gittens for the Treemap97 implementation of treemaps. We appreciate comments on a draft from Ben Bederson and Robert Allen.

REFERENCES

[AS94] Ahlberg, C., Shneiderman, B., “Visual information seeking: tight coupling of dynamic query filters with starfield displays”, *Proc. ACM CHI’94*, pp. 313-317, (1994).

[BC87] Becker, R., Cleveland, W., “Brushing scatterplots”, *Technometrics*, 29(2), pp. 127-142, (1987).

[BCS96] Buja, A., Cook, D., Swayne, D., “Interactive high-dimensional data visualization”, *Journal of Computational and Graphical Statistics*, 5(1), pp. 78-99, (1996).

[CBR97] Chi, E. H., Barry, P., Riedl, J., Konstan, J., “A spreadsheet approach to information visualization”, *Proc. IEEE Information Visualization ‘97*, pp. 17-24, (1997).

[CWM94] Chimera, R., Shneiderman B., “An exploratory evaluation of three interfaces for browsing large hierarchical tables of contents”, *ACM Transactions on Information Systems*, 12(4), pp. 383-406, (Oct. 94).

[DAP97] Dey, A., Abowd, G., Pinkerton, M., Wood, A., “CyberDesk: a framework for providing self-integrating ubiquitous software services”, *Proc. ACM UIST ‘97*, pp. 75-76, (1997).

[DP95] Dumas, J., Parsons, P., “Discovering the way programmers think about new programming environments”, *Communications of the ACM*, 38(6), pp. 45-56, (June 1995).

[ISB95] Isakowitz, T., Stohr, E., Balasubramanian, P., “RMM: a methodology for structured hypermedia design”, *Communications of the ACM*, 38(8), pp. 34-44, (August 1995).

[JBO94] Jacobson, A., Berkin, A., Orton, M., “LinkWinds: interactive scientific data analysis and visualization”, *Communications of the ACM*, 37(4), pp. 43-52, (April 1994).

[JS95] Jerding, D., Stasko, J., “The Information Mural:

a technique for displaying and navigating large information spaces", *Proc. IEEE Symposium on Information Visualization*, pp. 43-50, (October 1995).

[KS97] Kandogan, E., Shneiderman, B., "Elastic Windows: evaluation of multi-window operations", *Proc. ACM CHI'97*, pp. 250-257, (March 1997).

[LRB97] Livny, M., Ramakrishnan, R., Beyer, K., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J., Wenger, K., "DEVise: integrated querying and visual exploration of large datasets", *Proc. ACM SIGMOD'97*, pp. 301-312, (1997).

[MRC91] Mackinlay, J., Robertson, G., Card, S., "Perspective Wall: detail and context smoothly integrated", *Proc. ACM CHI'91*, pp. 173-179, (1991).

[NWS86] Norman, K., Weldon, L., Shneiderman, B., "Cognitive layouts of windows and multiple screens for user interfaces", *Intl Journal of Man-Machine Studies*, 25, pp. 229-248, (August 1986).

[NS97] North, C., Shneiderman, B., "A taxonomy of multiple window coordinations", University of Maryland, College Park, Dept of Computer Science Technical Report #CS-TR-3854, (1997).

[PCS95] Plaisant, C., Carr, D., Shneiderman, B., "Image browsers: taxonomy, guidelines, and informal specifications", *IEEE Software*, 12(2), pp. 21-32, (March 1995).

[RLS96] Roth, S., Lucas, P., Senn, J., Gomberg, C., Burks, M., Stroffolino, P., Kolojejchick, J., Dunmire, C., "Visage: a user interface environment for exploring information", *Proc. Information Visualization*, IEEE, pp. 3-12, (October 1996).

[Shn92] Shneiderman, B. "Tree visualization with treemaps: a 2-d space-filling approach", ACM Transactions on Graphics, 11(1), pp. 92-99, (Jan. 1992).

[SSS86] Shneiderman, B., Shafer, P., Simon, R., Weldon, L., "Display strategies for program browsing: concepts and an experiment", *IEEE Software*, 3(3), pp. 7-15, (March 1986).

[WA95] Ward, M., Allen, M., "High dimensional brushing for interactive exploration of multivariate data", *Proc. IEEE Visualization '95*, pp. 271-278, (1995).