Abstract

Title of Dissertation: A USER INTERFACE FOR COORDINATING VISUALIZATIONS BASED ON RELATIONAL SCHEMATA: SNAP-TOGETHER VISUALIZATION

Christopher Loy North, Doctor of Philosophy, 2000

Dissertation directed by: Professor Ben Shneiderman
Department of Computer Science

In the field of information visualization, researchers and developers have created many types of visualizations, or visual depictions of information. User interface designers often coordinate multiple visualizations, taking advantage of the strengths of each, to enable users to rapidly explore complex information. However, the combination of visualizations and coordinations needed in any given situation depends heavily on the data, tasks, and users. Consequently, the number of needed combinations explodes, and implementation becomes intractable.

Snap-Together Visualization (Snap) is a conceptual model, user interface, software architecture, and implemented system that enables users to rapidly and dynamically construct coordinated-visualization interfaces, customized for their data, without programming. Users load data into desired visualizations, then create coordinations between them, such as brushing and linking, overview and detail, and drill down.

This dissertation presents four primary contributions. First, Snap formalizes a conceptual model of visualization coordination that is based on the relational data model. Visualizations display relations, and coordinations tightly couple user interaction across relational joins.

Second, Snap's user interface enables the construction of coordinated-visualization interfaces without programming. Data users can dynamically mix and match visualizations and coordinations while exploring. Data disseminators can distribute appropriate interfaces with their data. Interface designers can rapidly prototype many alternatives.

Third, Snap's software architecture enables flexibility in data, visualizations, and coordinations. Visualization developers can easily snap-enable their independent visualizations using a simple API, allowing users to coordinate them with many other visualizations.

Fourth, empirical studies of Snap reveal benefits, cognitive issues, and usability concerns. Six data-savvy users successfully, enthusiastically, and rapidly designed powerful coordinated-visualization interfaces of their own. In a study with 18 subjects, an overview-and-detail coordination reliably improved user performance by 30-80% over detail-only and uncoordinated interfaces for most tasks.

Snap has proven useful in a variety of domains, including census statistics and geography, digital photo libraries, case-law documents, web-site logs, and traffic incident data.

A USER INTERFACE FOR COORDINATING VISUALIZATIONS BASED ON RELATIONAL SCHEMATA: SNAP-TOGETHER VISUALIZATION

by

Christopher Loy North

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2000

To
Jodi

For
Calli♥

Through
God

Acknowledgements

First, I thank Ben Shneiderman for advice, encouragement, and inspiration. Ben has been a wonderful advisor to me and major influence in my life. I could not possibly list all that I have learned from him, such as how to do research, give a talk, and work with others, as well as HCI content and the primacy of the human in the HCI equation. His encouraging words inspired me to continue when all seemed hopeless. Clearly, this dissertation would not exist without his influence.

Thanks to my dissertation committee members, Ben Bederson, Dave Mount, Kent Norman, and Adam Porter, for many suggestions that have improved this dissertation. Ben Bederson provided interesting discussions and reviews, and helped me learn to write a good TR. Kent Norman provided significant advice for the user studies. Bill Gasarch made things fun. Nancy Lindley was like a mom. Thanks also to Allan Kuchinsky for a thorough draft review.

I also thank all the members of the Human-Computer Interaction Lab whom I have been blessed to work with. Everything that HCIL produces is a group effort, so credit for this dissertation goes to them as well. Thanks to Catherine Plaisant, Anne Rose, Gary Marchionini, Eser Kandogan, Egemen Tanin, Zhijun Zhang, Flip Korn, and Ara, Rich, Ninad, Harsha, Khoa, Arkady, Teresa, Janet, Kathy, Jason, Steve, Rohit, Stephan, Dan, David, Anita, Laura, Tammara, Julia, Harry, Bob, Allison, Juan-Pablo, Lance, Hyunmo, and many others. I especially acknowledge Anna Fredrikson, Gunjan Dang, and Manav Kher for contributions to Snap. HCIL is a terrific group that I will miss.

I greatly thank those who funded this research, including the National Library of Medicine, NSF, UMIACS, WestLaw, Census Bureau, and Microsoft. Thanks to Kent Marquis at the Census Bureau for enabling the application of Snap to real problems. I thoroughly enjoyed working there with Rich, Lelyn, Betty, Laura, Dave Desjardins, Rob Creecy, and Tommy Wright. Thanks to Eric Malecki, Meds, and Michelle Baldonado at PARC for interesting diversions along the way.

Thanks also to my buddies at Montrose for support and prayers: Rich, John, Brian, Mango, Frank, Todd, Tπ, Hammerdog, Mark, Carl, Skoops, Gary, GReiff, and gang.

Special thanks to my family and family-in-law. They are who I am, and I will always learn more from them than can be written on paper. Thanks, Mom. To Cheri I owe life (or I'd still be chasing AI!). I thank my Dad whose footsteps I strive to follow. If I could be just half the man he is.

Alas, Jodi, by your sacrifice this dissertation has been achieved. This is your 'walk' too. I marvel at all you have accomplished during these years. In a mere nine months you accomplished more than I ever will.

Most of all, I thank God. This dissertation was directly inspired by Him. Without Him, I am nothing. Through Him, all things are possible. "For the foolishness of God is wiser than man's wisdom, and the weakness of God is stronger than man's strength." (1 Corinthians 1:25). Thank you, Lord Jesus.

∃LiJah wαS h∈Re .

♥*remember.*

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Problem

In the field of information visualization, researchers and developers have created many types of visualizations, or visual depictions of information [CMS99]. For example, to display hierarchical information, one can choose from outliners, Hyperbolic Trees [LR96], Treemaps [Shn92], fish-eye views [Fur86], etc. Each visualization has different strengths. For example, Hyperbolic Trees may be appropriate for deep unbalanced hierarchies, whereas Treemaps are helpful when nodes have numerical attributes (see Figure 1.1).

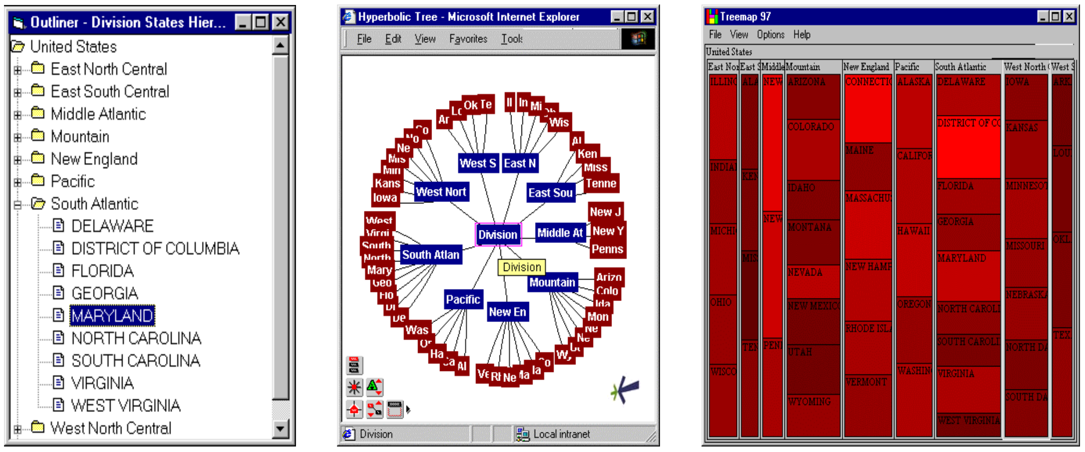


Figure 1.1: Hierarchy visualizations: Outliner, Hyperbolic Tree, Treemap

User interface designers often coordinate multiple visualizations, taking advantage of the strengths of each, to create even more powerful information exploration environments [Shn98] [BWK00]. This technique is particularly potent when the information is sufficiently complex to require different types of visualizations for different aspects or layers. A simple example interface is Microsoft's Windows Explorer (Figure 1.2), which employs 3 visualizations to browse hierarchical file systems: an outliner visualization of the folders, a tabular visualization of the files in the selected folder, and a textual visualization of the details of the selected file including a miniature quick-view.

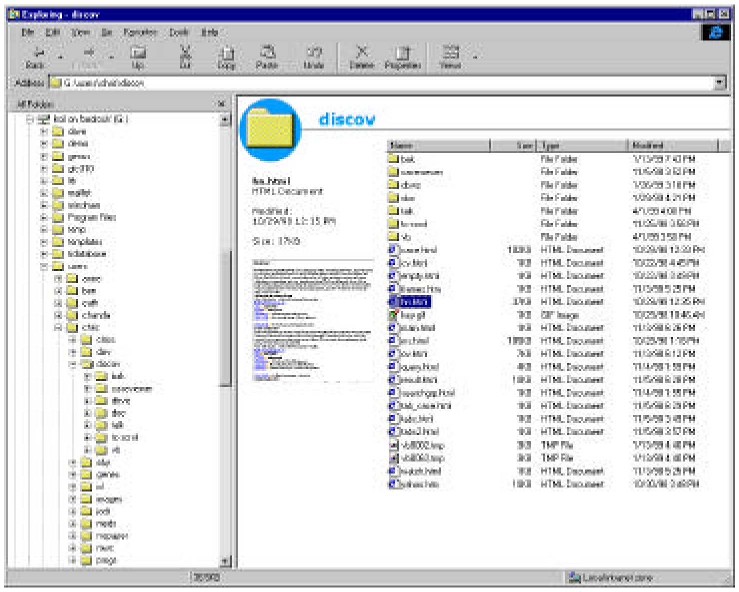


Figure 1.2: Windows Explorer, three coordinated visualizations

Visualizations can be coordinated in a variety of ways. In information-exploration interfaces, some common types of visualization coordinations [NS97] are:

- **Brushing and linking:** An exploratory data analysis (EDA) technique used when displaying a set of data items in multiple visualizations. When users select items in one visualization, those items are automatically highlighted in all the visualizations. A common example is brushing scatter plots [BC87]. For example, Figure 1.3 shows census data in Spotfire [AW95], a commercial data analysis package. Selecting the states with low percentages of high school and college graduates in the left plot reveals that those states also have low income and high unemployment levels in the plot on the right.

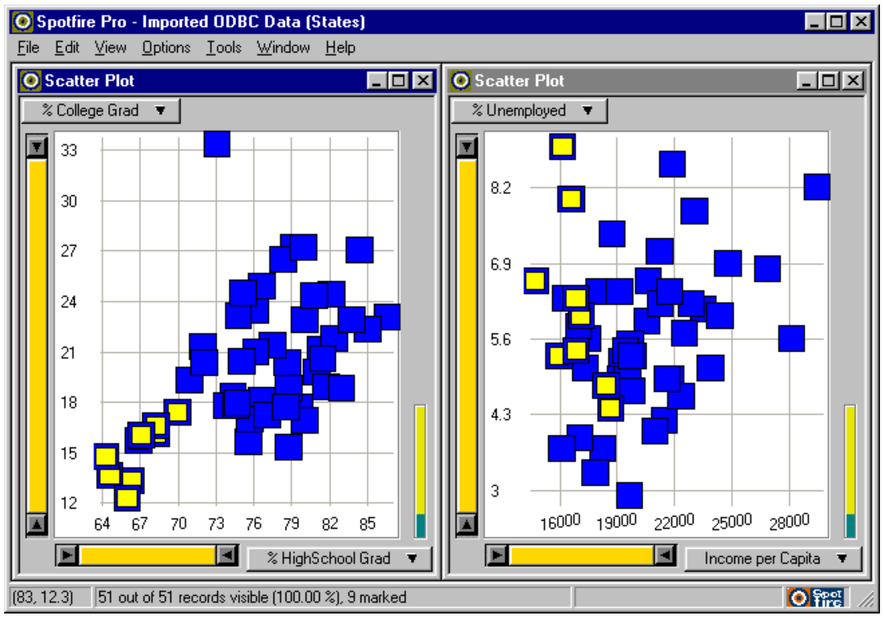
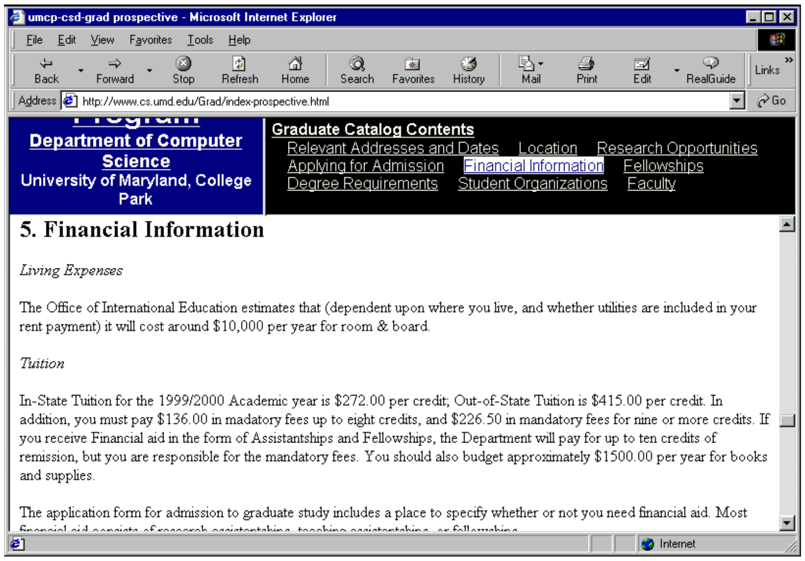


Figure 1.3: Brushing and linking in Spotfire

- **Overview and detail:** Selecting an item in the overview visualization navigates the detail visualization to the corresponding details. Items are represented visually smaller in the overview. This provides context and allows direct access to details. For example, web designers often add a table-of-contents frame to a large document. Users can then select a section title to scroll the main frame immediately to that section. In Figure 1.4, the user has selected the “Financial Information” section of the Graduate Catalog.


umcp-csd-grad prospective - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit RealGuide Links
Address http://www.cs.umd.edu/Grad/index-prospective.html
Go
Internet

Department of Computer Science
University of Maryland, College Park

Graduate Catalog Contents
Relevant Addresses and Dates Location Research Opportunities
Applying for Admission Financial Information Fellowships
Degree Requirements Student Organizations Faculty

5. Financial Information

Living Expenses

The Office of International Education estimates that (dependent upon where you live, and whether utilities are included in your rent payment) it will cost around $10,000 per year for room & board.

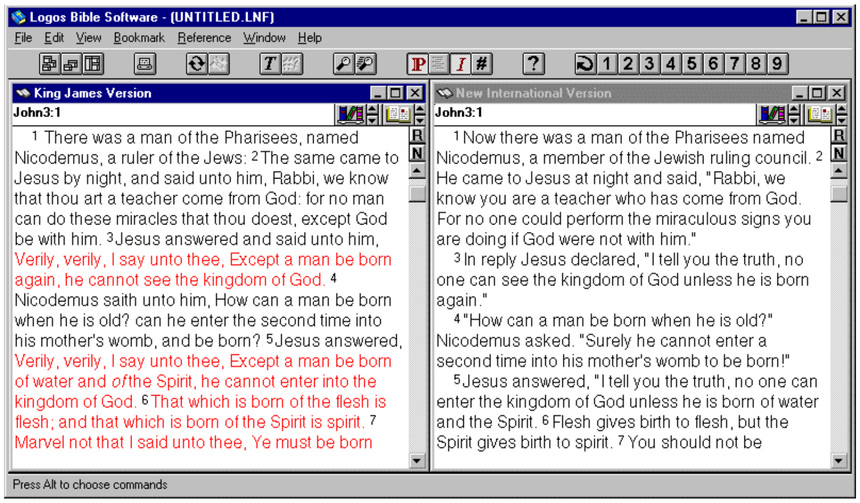
Tuition

In-State Tuition for the 1999/2000 Academic year is $272.00 per credit; Out-of-State Tuition is $415.00 per credit. In addition, you must pay $136.00 in madatory fees up to eight credits, and $226.50 in mandatory fees for nine or more credits. If you receive Financial aid in the form of Assistantships and Fellowships, the Department will pay for up to ten credits of remission, but you are responsible for the mandatory fees. You should also budget approximately $1500.00 per year for books and supplies.

The application form for admission to graduate study includes a place to specify whether or not you need financial aid. Most

Figure 1.4: Overview and detail with web frames

- **Drill down:** Allows users to navigate down successive layers of a hierarchical database. Selecting a parent item in one visualization loads children items into another visualization, as in Windows Explorer. This enables exploring very large-scale data, by displaying aggregates in one visualization and the contents of the selected aggregate in another visualization [FNP99].
- **Synchronized scrolling:** Users can conveniently scroll through multiple corresponding data sets. Examples include alternate translations, music, and information with summaries or annotations. In Figure 1.5, users of Logos Bible Software [Log93] can simultaneously scroll through multiple bible translations and commentaries by chapter and verse. This speeds users' tasks such as making comparisons or examining from multiple points of view.


Logos Bible Software - [UNTITLED.LNF]
File Edit View Bookmark Reference Window Help
1 2 3 4 5 6 7 8 9
King James Version
New International Version
R
N
Press Alt to choose commands

King James Version

John3:1

1 There was a man of the Pharisees, named Nicodemus, a ruler of the Jews: 2 The same came to Jesus by night, and said unto him, Rabbi, we know that thou art a teacher come from God: for no man can do these miracles that thou doest, except God be with him. 3 Jesus answered and said unto him, Verily, verily, I say unto thee, Except a man be born again, he cannot see the kingdom of God. 4 Nicodemus saith unto him, How can a man be born when he is old? can he enter the second time into his mother's womb, and be born? 5 Jesus answered, Verily, verily, I say unto thee, Except a man be born of water and *of* the Spirit, he cannot enter into the kingdom of God. 6 That which is born of the flesh is flesh; and that which is born of the Spirit is spirit. 7 Marvel not that I said unto thee, Ye must be born

New International Version

John3:1

1 Now there was a man of the Pharisees named Nicodemus, a member of the Jewish ruling council. 2 He came to Jesus at night and said, "Rabbi, we know you are a teacher who has come from God. For no one could perform the miraculous signs you are doing if God were not with him."

3 In reply Jesus declared, "I tell you the truth, no one can see the kingdom of God unless he is born again."

4 "How can a man be born when he is old?" Nicodemus asked. "Surely he cannot enter a second time into his mother's womb to be born!"

5 Jesus answered, "I tell you the truth, no one can enter the kingdom of God unless he is born of water and the Spirit. 6 Flesh gives birth to flesh, but the Spirit gives birth to spirit. 7 You should not be

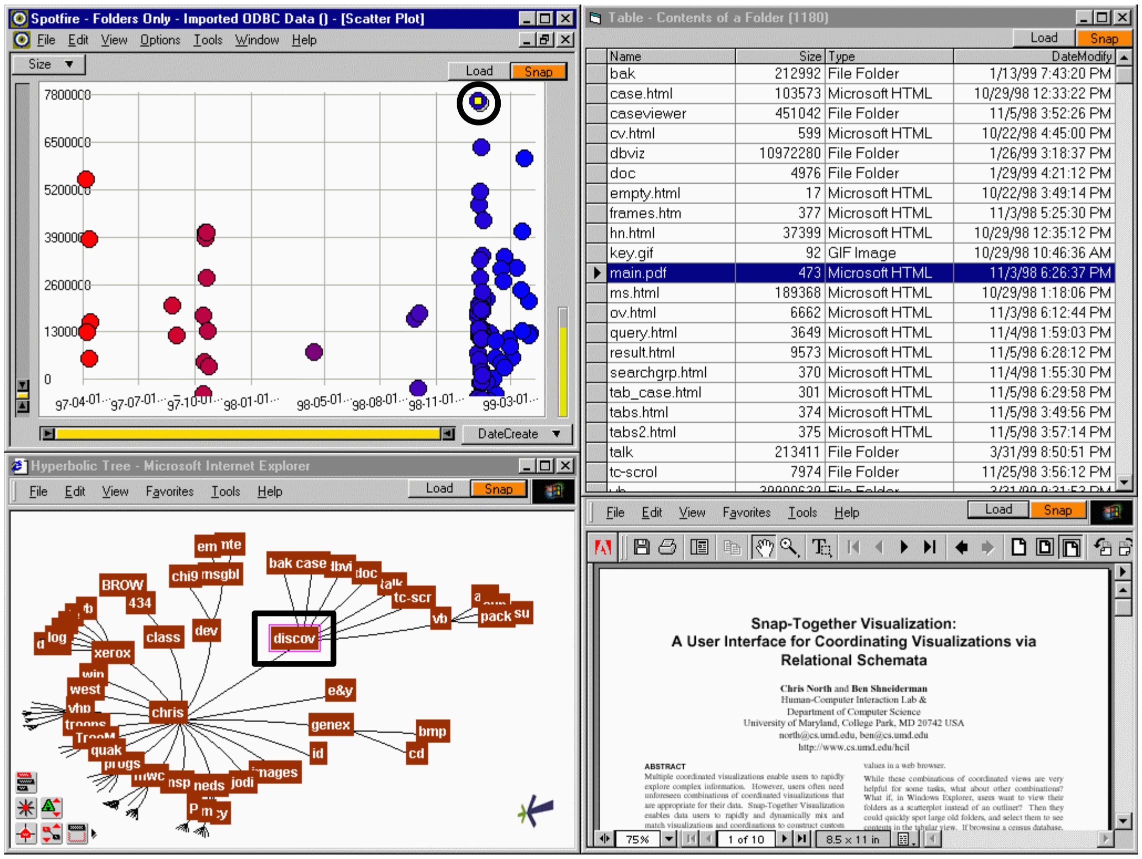
Figure 1.5: Synchronized scrolling with Logos Bible Software

A *coordinated-visualization* user interface is defined as a set of visualizations and a set of coordinations between the visualizations. In the literature, the phrase ‘multiple views’ is often used instead but sometimes refers strictly to the brushing-and-linking coordination. Hence, this dissertation uses ‘coordinated visualizations’ to refer to the more general definition and to reflect the focus of this research on visualization.

Many coordinated-visualization interfaces have been implemented. However, two confounding problems arise. First, the set of visualizations *and* coordinations needed in any given situation depends heavily on:

- **data:** different data sets have different features and structure.
- **tasks:** what does the user want to accomplish with the data?
- **users:** there is tremendous variation between users in individual user preferences, experience levels, etc.

For example, while Windows Explorer is helpful for some users and tasks, system administrators may need alternate visualizations. Replacing the outliner visualization of folders with a scatter plot of the folders would enable administrators to quickly spot large old folders for archival. In Figure 1.6, the scatterplot and hyperbolic tree display the folders, enabling users to examine size and date trends as well as hierarchical structure. Selecting a folder displays its files in the tabular visualization. Selecting a file displays its contents in the file viewer.



Name	Size	Type	DateModify
bak	212992	File Folder	1/13/99 7:43:20 PM
case.html	103573	Microsoft HTML	10/29/98 12:33:22 PM
caseviewer	451042	File Folder	11/5/98 3:52:26 PM
cv.html	599	Microsoft HTML	10/22/98 4:45:00 PM
dbviz	10972280	File Folder	1/26/99 3:18:37 PM
doc	4976	File Folder	1/29/99 4:21:12 PM
empty.html	17	Microsoft HTML	10/22/98 3:49:14 PM
frames.htm	377	Microsoft HTML	11/3/98 5:25:30 PM
hn.html	37399	Microsoft HTML	10/29/98 12:35:12 PM
key.gif	92	GIF Image	10/29/98 10:46:36 AM
main.pdf	473	Microsoft HTML	11/3/98 6:26:37 PM
ms.html	189368	Microsoft HTML	10/29/98 1:18:06 PM
ov.html	6662	Microsoft HTML	11/3/98 6:12:44 PM
query.html	3649	Microsoft HTML	11/4/98 1:59:03 PM
result.html	9573	Microsoft HTML	11/5/98 6:28:12 PM
searchgrp.html	370	Microsoft HTML	11/4/98 1:55:30 PM
tab_case.html	301	Microsoft HTML	11/5/98 6:29:58 PM
tabs.html	374	Microsoft HTML	11/5/98 3:49:56 PM
tabs2.html	375	Microsoft HTML	11/5/98 3:57:14 PM
talk	213411	File Folder	3/31/99 8:50:51 PM
tc-scrol	7974	File Folder	11/25/98 3:56:12 PM

Snap-Together Visualization:
A User Interface for Coordinating Visualizations via Relational Schemata

Chris North and Ben Shneiderman
Human-Computer Interaction Lab &
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
Multiple coordinated visualizations enable users to rapidly explore complex information. However, users often need unforeseen combinations of coordinated visualizations that are appropriate for their data. Snap-Together Visualization enables data users to rapidly and dynamically mix and match visualizations and coordinations to construct custom

values in a web browser.

While these combinations of coordinated views are very helpful for some tasks, what about other combinations? What if, in Windows Explorer, users want to view their folders as a scatterplot instead of an outliner? Then they could quickly spot large old folders, and select them to see contents in the tabular view. If browsing a census database,

75% 1 of 10 8.5 x 11 in

Figure 1.6: A coordinated-visualization interface for browsing folders and files.

Secondly, the implemented visualizations are typically not programmed to coordinate together. Hence, these alternate combinations usually require custom development. Researchers stumble over this problem often, and must constantly re-implement coordinations between new unforeseen combinations of visualizations. Unfortunately, this is a poor solution to the problem. Even with good component-based design, these hard-coded combinations are inflexible and difficult to construct.

Clearly, the number of needed combinations of visualizations and coordinations explodes exponentially, and implementation becomes intractable. Hence, the control of the choice of coordinated-visualization interface needs to be in the hands of the users. A lightweight mechanism is needed to allow end-users to easily "snap" individual visualizations together into custom combinations. This must not be a toolkit that requires programming, but a user interface.

1.2 Snap-Together Visualization

Snap-Together Visualization (Snap) [NS00a] is a conceptual model, user interface, architecture, and implemented system developed to meet these needs. Snap enables data users to rapidly and dynamically mix and match visualizations and coordinations to construct custom exploration interfaces without programming. Snap is flexible in data, visualizations and coordinations. Snap focuses on (a) interconnecting the visualization tools created by researchers and developers in the field to (b) construct coordinated-visualization interfaces for rapid exploration and navigation of data and relationships.

Snap is based on the relational data model. To explore a database, users first display relations (tables or query results) in visualizations. Then they construct coordinations by specifying actions to tightly couple between the visualizations. Visualization developers can easily make their independent visualizations snap-able using a simple API.

1.2.1 Scenario

This scenario demonstrates step-by-step how Snap is used to construct the file-folder browser for system administrators as described in the example in the introduction (see [NS99] video for dynamic interaction). First, the database containing the folder and file information is opened with Snap. The Snap Menu window (Figure 1.7) displays the list of tables and queries in the database (left), as well as a list of available visualization types (right). To view the folders in a Spotfire scatter plot, the table containing folder information is dragged from the list and dropped onto the scatter plot button. The plot opens, loads, and displays the folders. Choosing ‘creation date’ for the X-axis and ‘size’ for the Y-axis establishes the desired visualization. Now it is easy to spot the large old folders in the upper left of the plot. Of course, users need to see the files contained in the folders. Dragging the query that extracts only the files within a given folder, and dropping it onto the tabular visualization button opens the new visualization.

Each visualization window is adorned with a snap button Snap. To coordinate the visualizations, the snap button is dragged from the plot to the tabular visualization (Figure 1.8). The Snap Specification dialog (Figure 1.9) then displays the available actions in each visualization that can be tightly coupled. Choosing the ‘select’ action in the plot and the ‘load’ action in the tabular visualization specifies that selecting a folder in the plot should load and display the files in that folder into the tabular visualization.

Now, construction of the coordinated-visualization interface is complete (Figure 1.10). Users can browse by simply selecting folders in the plot and viewing contents in the tabular visualization, like Windows Explorer.

Additional visualizations could be added to further improve the interface (as in Figure 1.6). For example, if the context of the folders in the hierarchical structure is important, then users might load the folders into Inxight's Hyperbolic Tree. They could coordinate this to the scatter plot so that selecting a folder in either visualization would also select and highlight it in the other. To examine the contents of many files, users could coordinate a file viewer onto the tabular visualization.

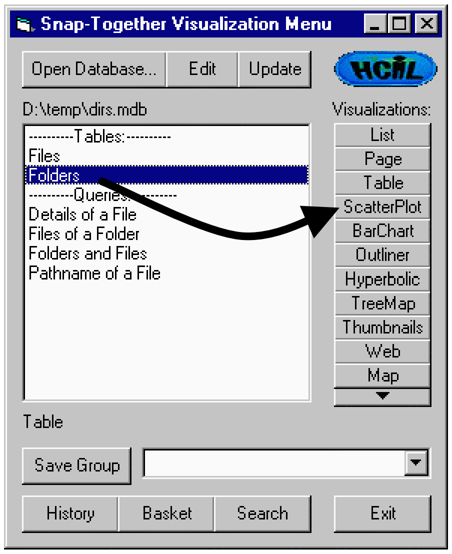
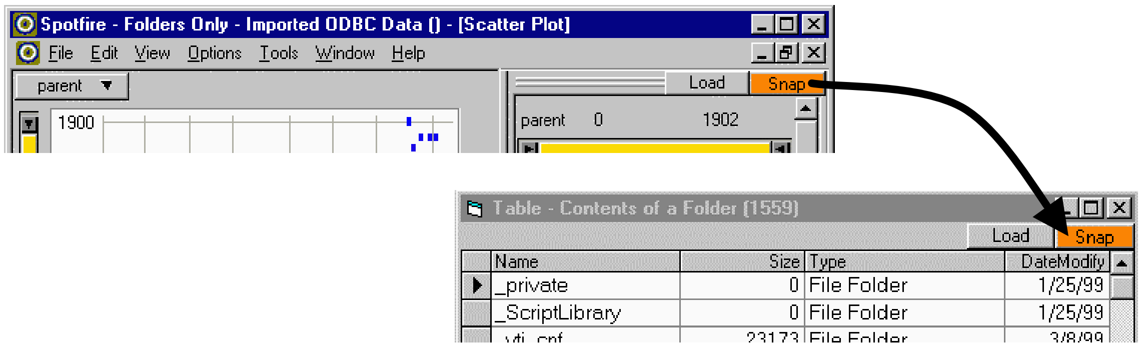
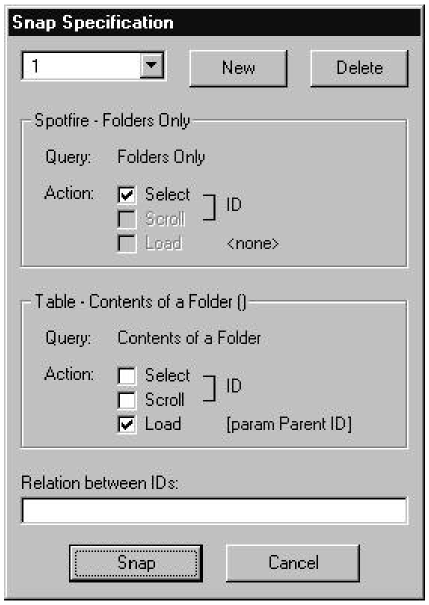


Figure 1.7: Opening visualizations


Spotfire - Folders Only - Imported ODBC Data () - [Scatter Plot]
File
Edit
View
Options
Tools
Window
Help
parent
1900
Load
Snap
parent 0 1902
Table - Contents of a Folder (1559)
Load
Snap

Name	Size	Type	DateModify
_private	0	File Folder	1/25/99
_ScriptLibrary	0	File Folder	1/25/99
_vti_cnf	23173	File Folder	3/8/99

Figure 1.8: Coordinating visualizations



Snap Specification

1 | New | Delete

Spotfire - Folders Only

Query: Folders Only

Action: [✓ Select] Scroll Load ID <none>

Table - Contents of a Folder ()

Query: Contents of a Folder

Action: Select Scroll [✓ Load] ID [param Parent ID]

Relation between IDs:

Snap | Cancel

Figure 1.9: Specifying the coordination

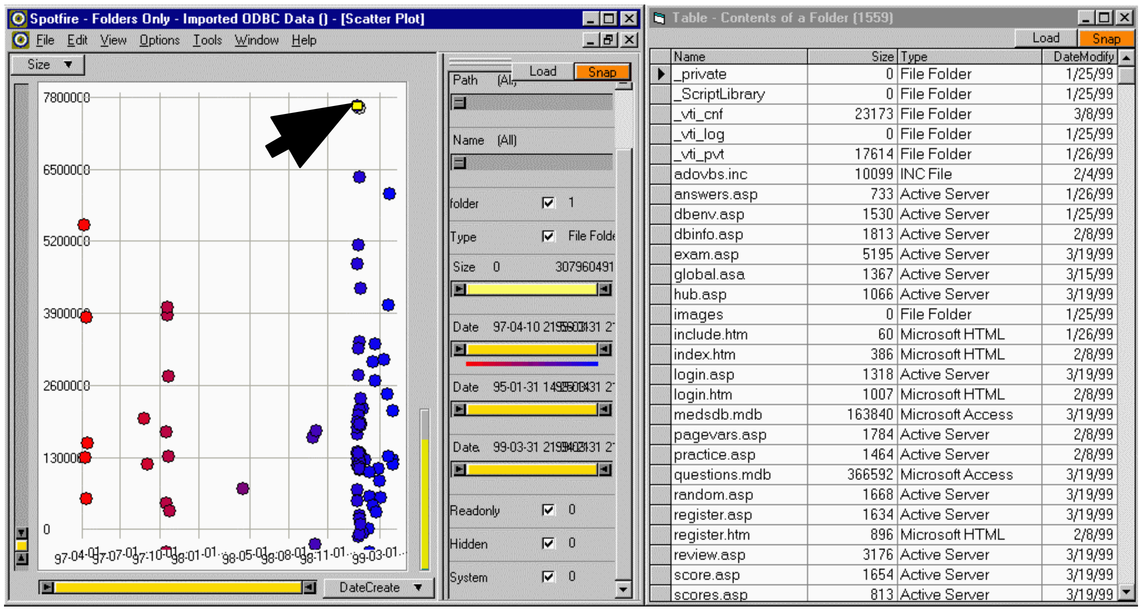


Table - Contents of a Folder (1559)

Load | Snap

Name	Size	Type	DateModify
_private	0	File Folder	1/25/99
_ScriptLibrary	0	File Folder	1/25/99
_vti_cnf	23173	File Folder	3/8/99
_vti_log	0	File Folder	1/25/99
_vti_pvt	17614	File Folder	1/26/99
adovbs.inc	10099	INC File	2/4/99
answers.asp	733	Active Server	1/26/99
dbenv.asp	1530	Active Server	1/25/99
dbinfo.asp	1813	Active Server	2/8/99
exam.asp	5195	Active Server	3/19/99
global.asa	1367	Active Server	3/15/99
hub.asp	1066	Active Server	3/19/99
images	0	File Folder	1/25/99
include.htm	60	Microsoft HTML	1/26/99
index.htm	386	Microsoft HTML	2/8/99
login.asp	1318	Active Server	3/19/99
login.htm	1007	Microsoft HTML	2/8/99
medsdb.mdb	163840	Microsoft Access	3/19/99
pagevars.asp	1784	Active Server	2/8/99
practice.asp	1464	Active Server	2/8/99
questions.mdb	366592	Microsoft Access	3/19/99
random.asp	1668	Active Server	3/19/99
register.asp	1634	Active Server	3/19/99
register.htm	896	Microsoft HTML	2/8/99
review.asp	3176	Active Server	3/19/99
score.asp	1654	Active Server	3/19/99
scores.asp	813	Active Server	3/19/99

Figure 1.10: Operating the constructed interface

1.3 Research Questions

In providing Snap as a solution to the stated problem, this research must provide answers to several important questions.

First, the concept of visualization coordination is not well understood. Coordination has been only loosely viewed as a form of interaction. There has been no categorization of types of coordination, nor formal theory of coordination. Already the above definition of a coordinated-visualization interface is a significant advance in understanding the concept. How can visualization coordination be formally modeled?

Second, how can end-users construct their own coordinated-visualization interfaces without programming? What user interface will enable them to accomplish this?

Third, how can the software architecture provide such flexibility, and enable the use of independent visualization tools developed by others? The effort required by visualization developers to enable their tools must be minimized, while maximizing the functionality available to users.

Finally, empirical evaluation is needed to understand users ability to construct and operate coordinated visualizations. Do users understand coordination between visualizations? Can they construct their own coordinated-visualization environments to support their tasks? Can they use it to their benefit? If there is a benefit, why and what are the critical aspects of the coordinated visualizations that causes improvement? In general, user interface design requires significant expertise, but Snap puts some design capability in the hands of users. Can users essentially design their own user interfaces for information exploration by snapping together appropriate visualizations?

1.4 Scope

Snap focuses on coordinations between visualizations for information exploration. Snap does not address other classes of tasks such as data input or editing.

Snap focuses on the common types of data and visualizations that are typically encountered in the field of information visualization, such as databases, file directories, statistical tables, etc. It is less concerned with scientific visualization applications which are often more oriented towards image processing.

Snap focuses on common coordinations for information exploration. There are other kinds of coordination for data manipulation consistency, dynamic data, collaboration, etc.

1.5 Content

Chapter 2 reviews related literature, and provides a framework of the space that Snap fits within. Chapter 3 describes Snap's foundational model of visualization coordination. Chapter 4 describes Snap's user interface for coordination construction. Chapter 5 describes Snap's architecture that enables independent visualization tools. Chapter 6 details two empirical studies of coordination construction and operation. Chapter 7 concludes with benefits, limitations, contributions, and future work. Appendix A demonstrates Snap with several additional scenarios to show its breadth and usefulness. The file-folders scenario presented above (Figure 1.6) is used throughout the dissertation for examples.

Chapter 2:
Related Work

2.1 Conceptual Models of Coordination

Previous work on multiple window strategies [Shn98], [NWS86], [SSS86], [WH87], [CPF84], [Woo84] have loosely characterized a few examples of coordination. In statistical graphics, the brushing-and-linking coordination has been formally defined [BC87] and software architectures specified [MSB90]. In general, these systems add a color attribute to the underlying data records. Brushing a data point modifies the color attribute of its record, and affects its display in other plots. In the image-browsing domain, the overview-and-detail coordination has been formally defined [PCS95] using constraints between a field-of-view box in the overview and the panning scroll bars in the detail (Figure 2.1).

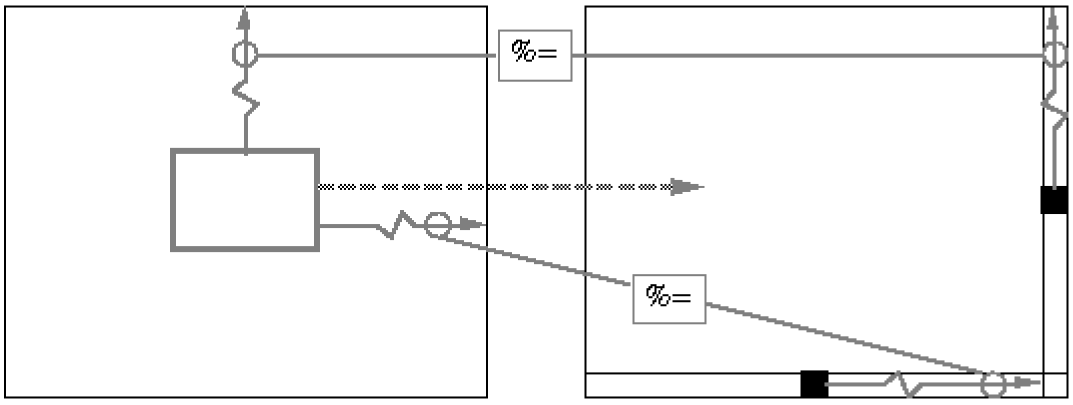


Figure 2.1: Overview and detail specification for image browsing

2.1.1 Object-based vs. Attribute-based Coordination

In general, coordination in information visualization can operate as either object-based or attribute-based. In object-based coordination, users interact with individual data objects such as folders and files, data points, etc. Brushing and linking data points is an example. Yet, while object-based coordination is the more common form of interaction, no formal model comprehensively describes the behavior of the common coordinations in the object-based approach.

In attribute-based coordination, users interact with the attribute space containing the data objects. This has two primary uses: spatial navigation and filtering. Spatial navigation in 2D and 3D spaces is used in coordinating the panning and zooming of data plots with common axes and image browsers, based on attribute ranges. Filtering, as in Dynamic Queries [AS94], enables selection or elimination of data points by specifying attribute ranges in queries. These applications have been well specified.

Snap focuses on object-based coordination, because it is the more common form of interaction and of more general utility to information visualization, and has not been well explored. Object-based coordination is more general in terms of supporting many different types of data and visualizations.

2.2 Flexibility in Coordinated Visualization

Systems with coordinated-visualization user interfaces can be classified by their level of flexibility in data, visualizations, and coordinations:

1. **Data flexible**: users can load their own different data sets into the visualizations.

2. **Visualization flexible**: users can choose different sets of visualizations as appropriate for the data.
3. **Coordination flexible**: users can choose different types of coordinations between pairs of visualizations as needed for exploring or navigating relationships in the data.

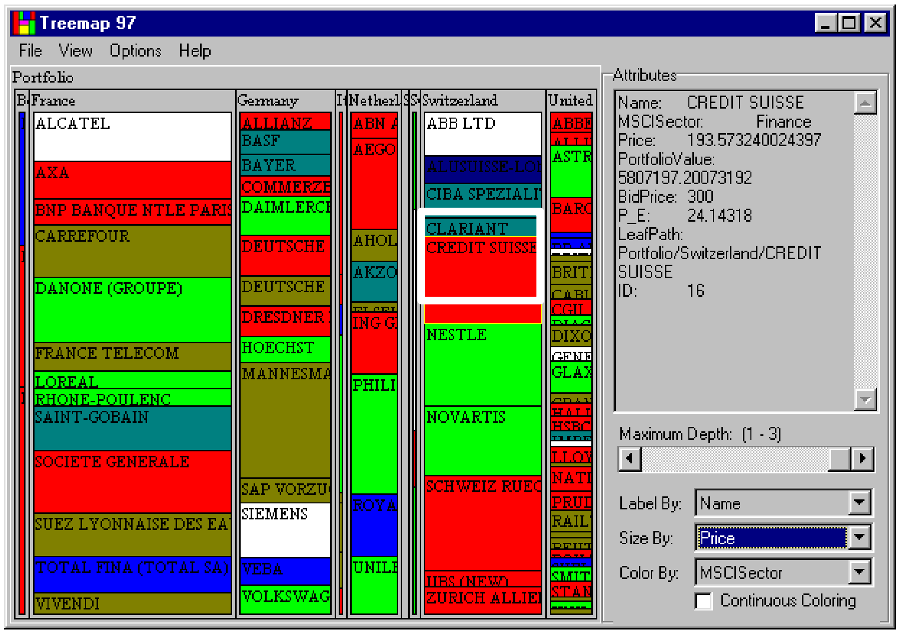
Some systems are not intended for flexibility. For example, Windows Explorer always displays the same data set (the hard drive file structure), with the same visualizations and coordinations.

2.2.1 Data Flexible

Most systems are at the first level of flexibility. They are flexible for data but not for visualizations or coordinations. Users can load their own data, but are always presented with the same hard-coded coordinated-visualization interface.

For example, the Treemap visualization tool (Figure 2.2) can load and display any hierarchical data set of users' choosing, but remains constant in its pair of visualizations (the Treemap visualization and the details pane) and the coordination between them (selecting a node in the Treemap displays associated data in the details pane).

Many data-flexible systems have been implemented, covering a variety of domains. Appendix B provides a taxonomy of these systems, based on the types of coordinations they use, with descriptions of many of the systems.



Name: CREDIT SUISSE
MSCISector: Finance
Price: 193.573240024397
PortfolioValue:
5807197.20073192
BidPrice: 300
P_E: 24.14318
LeafPath:
Portfolio/Switzerland/CREDIT SUISSE
ID: 16

Label By: Name
Size By: Price
Color By: MSCISector

Figure 2.2: Treemap (left), and details pane (top right)

2.2.2 Visualization Flexible

At the second level of flexibility, systems are flexible in choice of visualizations (and data). However, users cannot establish a different type of coordination between two visualizations with these systems.

Exploratory data analysis (EDA) systems, such as Datadesk [Vel88], SAS Insight/JMP, EDV [EW95] (Figure 2.3), and Spotfire [AW95], display a data table in many different types of visualizations of users' choosing such as scatter plots, bar charts or histograms. All the visualizations are coordinated for brushing-and-linking, allowing users to relate data points across visualizations. These systems provide a toolbox of visualizations that users can choose from (as in Figure 2.4). In each of these systems, the brushing-and-linking coordination is a fixed and global operation in their interfaces.

Some systems such XGobi [BCS96] let users specify many options for the brushing, such as accumulation, color, glyphs, etc.

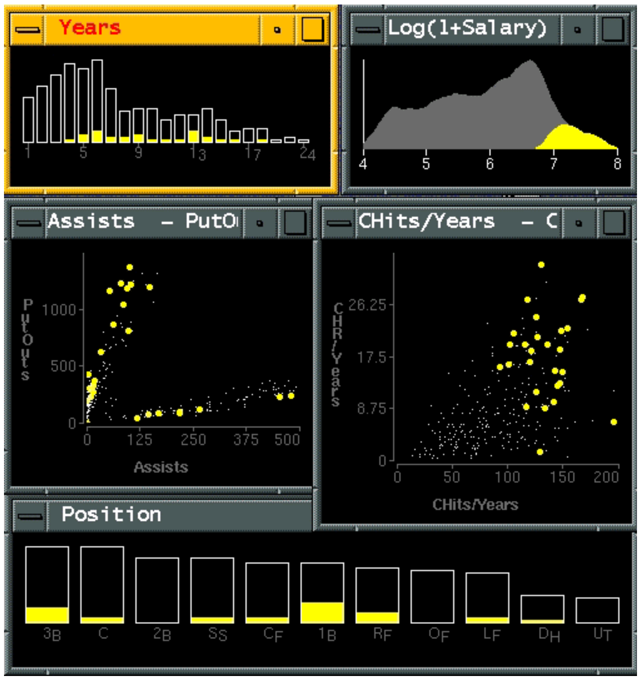


Figure 2.3: EDV, brushing and linking

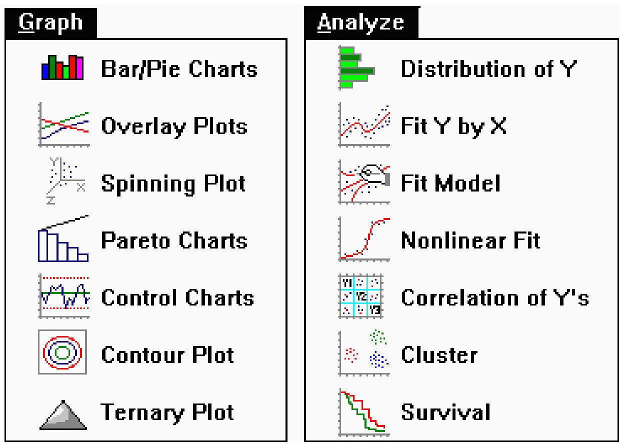


Figure 2.4: SAS JMP visualization toolbox menus

In databases, Visage [RLS96] extends the brushing coordination to multiple tables by brushing across relational joins. With Visage's "information-centric" approach, users can drag-and-drop data items between visualizations to display them in different ways. The Visage VQE [DRK97] component also coordinates dynamic queries across all visualizations within a VQE window. The Visage SAGE component (Figure 2.5) generates different types of visualizations. Users specify the visualization by associating data attributes with visual elements.

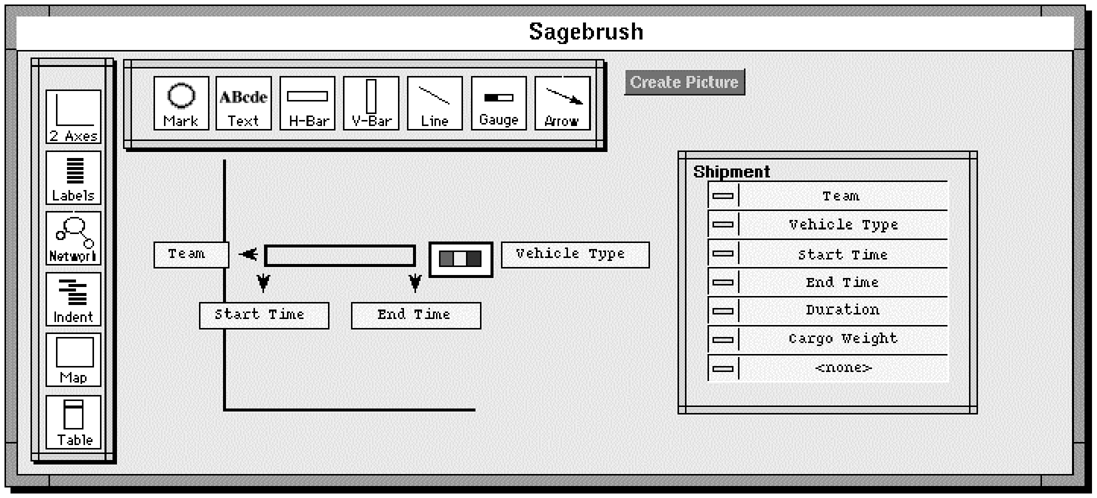
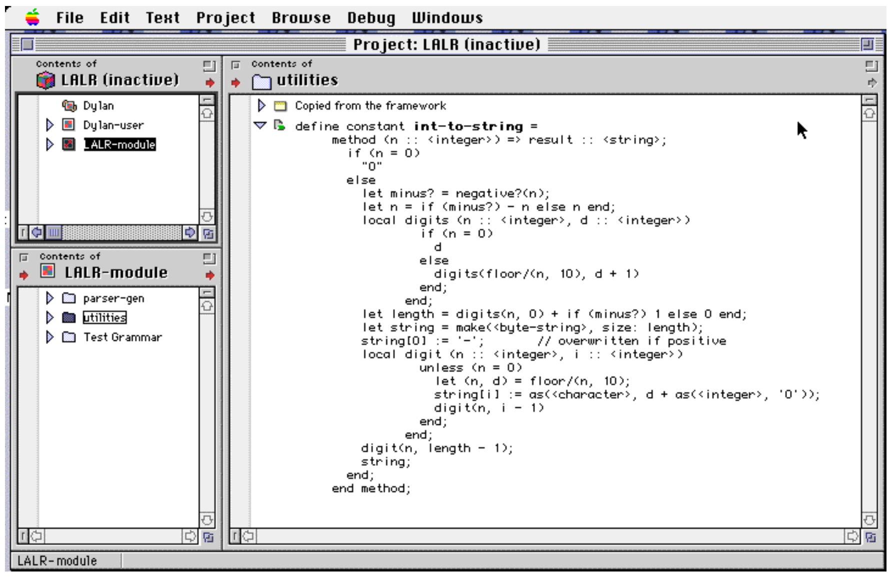


Figure 2.5: Visage's SAGE specifying a horizontal bar chart

2.2.3 Coordination Flexible

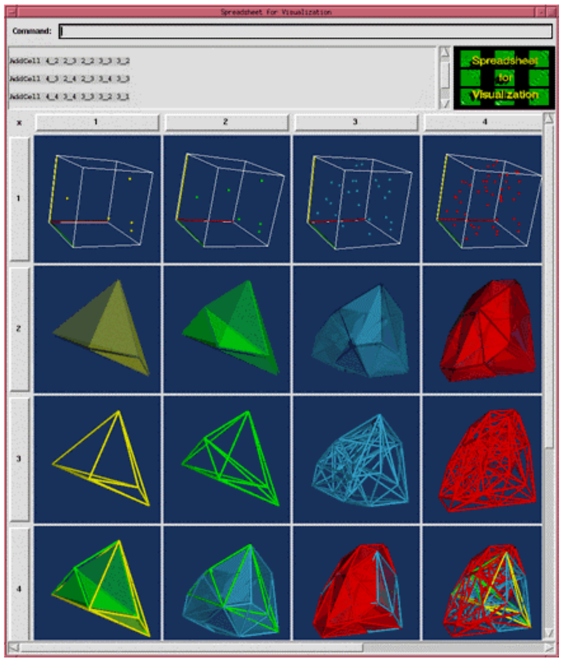
At the third level of flexibility, systems are flexible in the coordinations between visualizations (and generally flexible in data and visualizations too). There are two kinds of flexibility in coordination: choosing the visualizations to coordinate, and specifying the type of coordination between them.

Most of these systems provide only one type of coordination but let users choose which visualizations to coordinate. The Apple Dylan programming environment [DP95] (Figure 2.6) lets users browse hierarchical object-oriented programs by splitting and linking frames so that selecting a folder in one frame displays its contents in the other frame (e.g. generalized Windows Explorer). To link frames, users drag the ‘output arrow’ from one frame to the ‘input arrow’ of another frame. Spreadsheet Visualization [CBR97] (Figure 2.7) arranges many small 3D visualizations as cells in a 2D grid. Then, users can select a whole row or column of visualizations to synchronize their 3D navigation. With Logos Bible Software, users can coordinate scrolling text windows of different translations and commentaries to synchronize scroll based on chapter and verse. Users select from a window list to synchronize one window to another (Figure 2.8).



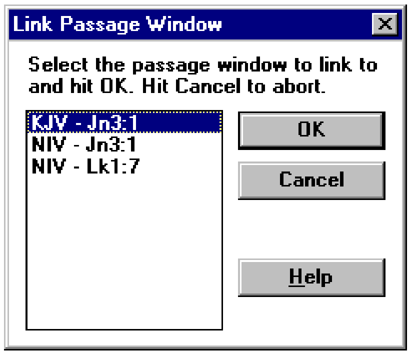
```
define constant int-to-string =
      method (n :: <integer>) => result :: <string>;
        if (n = 0)
          "0"
        else
          let minus? = negative?(n);
          let n = if (minus?) - n else n end;
          local digits (n :: <integer>, d :: <integer>)
                  if (n = 0)
                    d
                  else
                    digits(floor/(n, 10), d + 1)
                  end;
                end;
          let length = digits(n, 0) + if (minus?) 1 else 0 end;
          let string = make(<byte-string>, size: length);
          string[0] := '-';       // overwritten if positive
          local digit (n :: <integer>, i :: <integer>)
                  unless (n = 0)
                    let (n, d) = floor/(n, 10);
                    string[i] := as(<character>, d + as(<integer>, '0'));
                    digit(n, i - 1)
                  end;
                end;
          digit(n, length - 1);
          string;
        end;
      end method;
```

Figure 2.6: Apple Dylan with three split and linked frames



```
AddCell 4_2 2_3 2_2 3_3 3_2
AddCell 4_3 2_4 2_3 3_4 3_3
AddCell 4_4 3_4 3_3 3_2 3_1
```

Figure 2.7: Spreadsheet Visualization



Link Passage Window

Select the passage window to link to and hit OK. Hit Cancel to abort.

KJV - Jn3:1
NIV - Jn3:1
NIV - Lk1:7

Figure 2.8: Logos dialog for choosing windows to synchronize scroll

DEVise [LRB97] allows users to select some different types of coordinations between visualizations. In plots with common axes, users can synchronize panning and zooming between plots or create a field-of-view box in one plot to control another (Figure 2.9, Figure 2.10). Users can also establish set operations between visualizations so that the data in several visualizations can be combined and displayed in another visualization. Various menus and dialog boxes are used to establish these coordinations. It is interesting that the mechanisms for establishing each type of coordination are very different in the DEVise user interface. As in Visage, users create visualizations by mapping data attributes to visual elements.

In the image-browsing domain, LinkKit [Nor98] (Figure 2.11) allows users to display and coordinate different 2D views of the Visible Human 3D image data. Users can coordinate views for orthogonal slicing, synchronized slicing, and panning by field-of-view box.

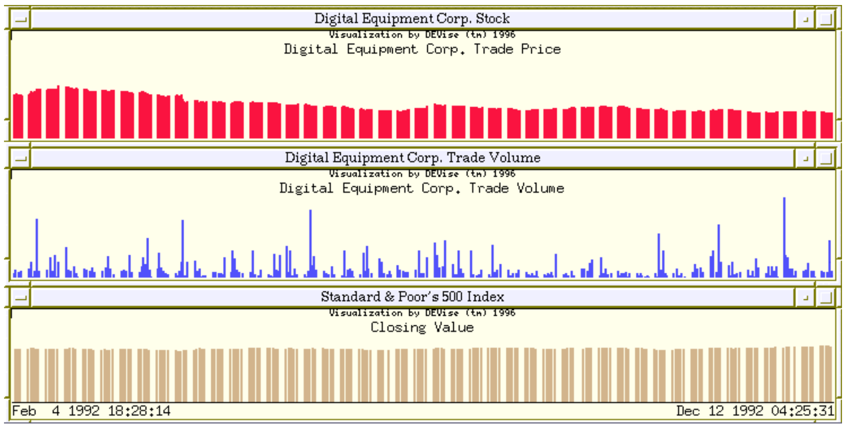
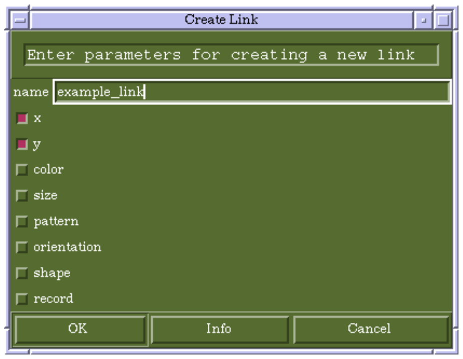


Figure 2.9:   DEVise, three bar charts synchronized by date on the X-axis



Enter parameters for creating a new link

name: example_link

- [✓] x
- [✓] y
- [ ] color
- [ ] size
- [ ] pattern
- [ ] orientation
- [ ] shape
- [ ] record

Figure 2.10: DEVise dialog for specifying plot attributes to synchronize

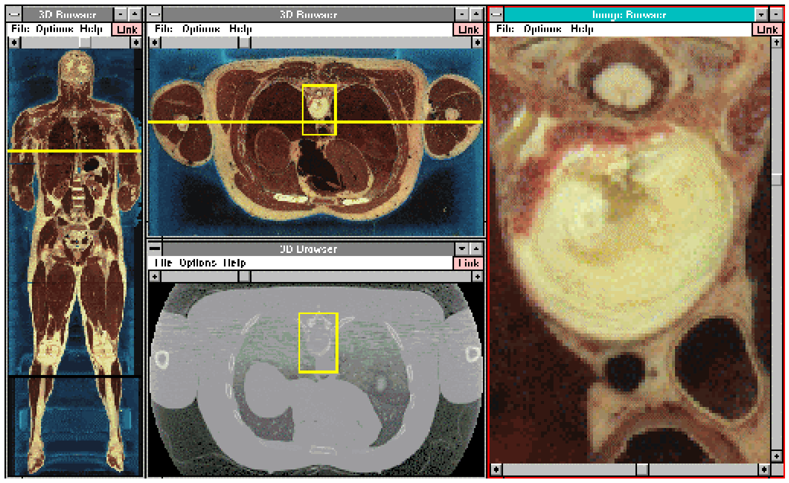


Figure 2.11: LinkKit navigating the Visible Human

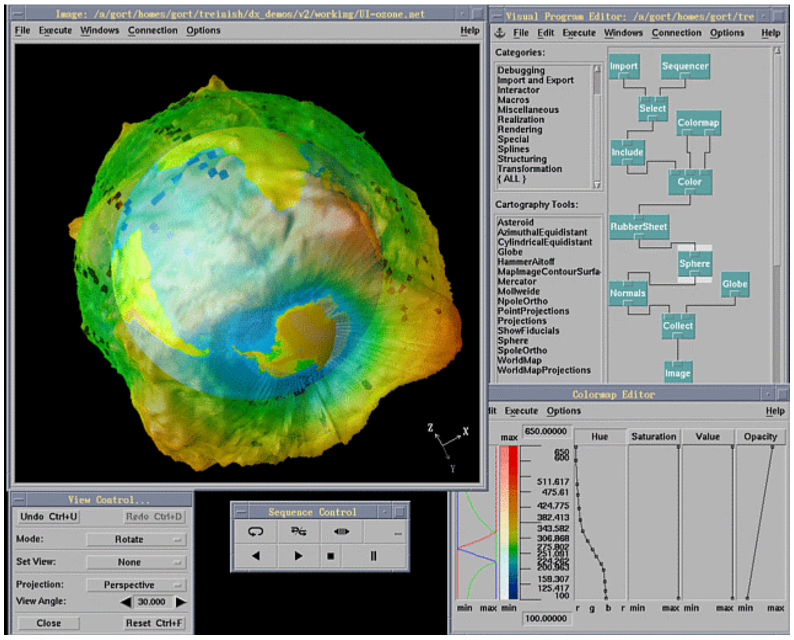
Snap builds on these systems. It borrows Visage's information-centric approach (object-based), making individual information items the basis of coordination rather than 2D information-space axes as in DEVise or LinkKit (attribute-based). Snap uses a drag-and-drop action similar to Apple Dylan to select visualizations to coordinate. However, Snap's coordination model, specification user interface, software architecture and ultimate purpose are unique. Snap allows users to construct a variety of common coordinations quickly and easily.

Snap also differs in its use of independent visualizations. Each of these systems uses a fully integrated architecture, in which visualizations are implemented within the system itself. Snap's architectural approach is similar to that of the Cyberdesk prototype [DAP97], which allows users to select text in any window and then choose a "service", such as a web search or address book application, to display search hits for that text. Independent applications can easily register themselves as an available service.

2.3 Construction in Visualization

There are a variety of other approaches used for construction in visualization environments.

In scientific visualization, data-flow systems such as ConMan [Hae88], AVS, and IBM Data Explorer (Figure 2.12), also employ a form of dynamic linking, but for a different purpose. Users link a variety of modules to create custom data processing and visualization pipelines, much like pipes on the Unix command line. Complex data structures are passed between modules. Some modules computationally transform the data before passing it on, and some display the data graphically. In contrast, Snap focuses on coordinating user interaction in visualizations. Snap coordinations transmit interaction rather than data, and coordinations are bi-directional like constraints rather than pipes.

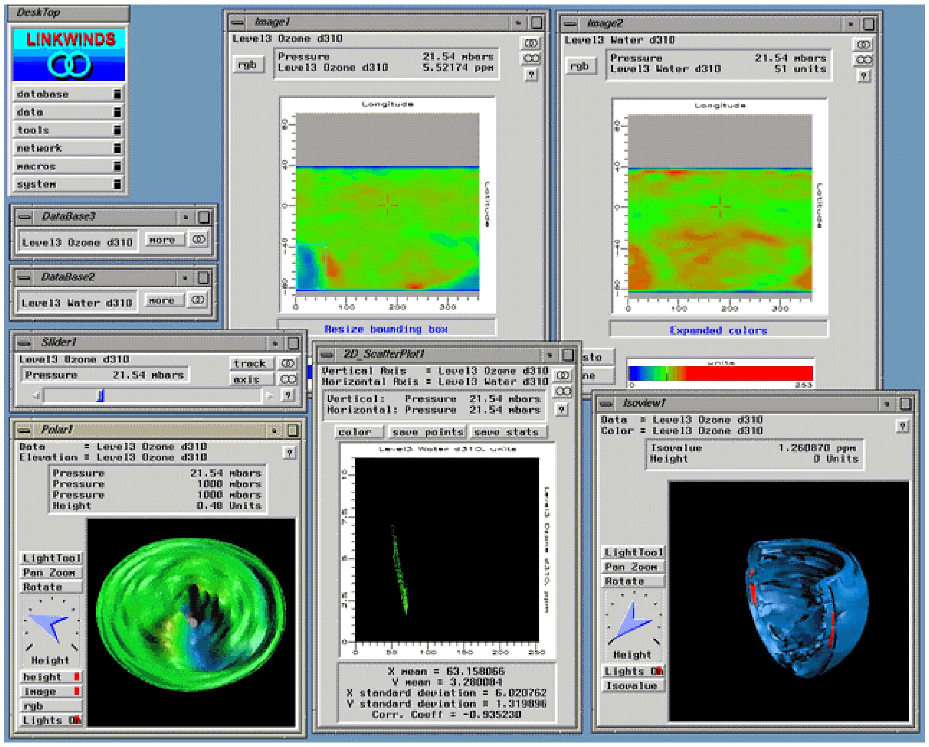


Mode: Rotate
Set View: None
Projection: Perspective
View Angle: 30.000

max 650.00000
min 100.00000

Figure 2.12: IBM Data Explorer, data-flow (right) and visualization (left)

Filter-flow systems such as Linkwinds [JBO94] (Figure 2.13) behave similar to data-flow systems, but provide interactive data filtering capability. Users link dynamic query filter controls and visualizations in a pipeline network. Selecting attribute ranges in a control or visualization filters the data displayed downstream in the pipeline.



```
Level3 Ozone d310
Level3 Water d310

Level3 Ozone d310
Pressure          21.54 mbars

Data      = Level3 Ozone d310
Elevation = Level3 Ozone d310
Pressure                 21.54 mbars
Pressure                  1000 mbars
Pressure                  1000 mbars
Height                    0.48 Units

Level3 Ozone d310
Pressure                 21.54 mbars
Level3 Ozone d310       5.52174 ppm

Level3 Water d310
Pressure                 21.54 mbars
Level3 Water d310          51 units

Vertical Axis   = Level3 Ozone d310
Horizontal Axis = Level3 Water d310
Vertical:    Pressure   21.54 mbars
Horizontal:  Pressure   21.54 mbars

          X mean = 63.158066
          Y mean = 3.280084
X standard deviation = 6.020762
Y standard deviation = 1.319896
        Corr. Coeff = -0.935230

Data  = Level3 Ozone d310
Color = Level3 Ozone d310
Isovalue                1.260870 ppm
Height                     0 Units
```

Figure 2.13: LinkWinds

Constraint-based tools allow users to construct interactive displays by specifying various mathematical relationships between objects. These systems are generally intended for specification of more complex interaction within a visualization. For example, with ThingLab [Bor86] users can construct complex interfaces that respond to direct manipulation interaction. [HM90] provides a simplified constraint-based specification for visual layout of objects in a user interface that adjusts to resizing.

LiveDocs [MHG00] and InfoStill [CHH99] allow authors to easily publish visualizations as interactive data reports. Authors can place a few different types of data plots (coordinated for brushing and linking) within context on a web page, and use hypertext links to invoke various saved states of the visualizations.

At the opposite end of the spectrum from Snap are visualization programming toolkits. Toolkits provide programmers with a library of reusable visualization primitives. However, coordination beyond brushing-and-linking is rarely included as a primitive. Amulet [MMM97] includes constraint capabilities that can be helpful for implementing coordinations, but are still at the programmer level. Technologies such as COM and CORBA [Vin97] are improving programmers' capability to establish communication between independent applications, a key ingredient for coordinating independent visualizations.

2.4 Evaluation

Little work has been done to study and evaluate the use of coordinated visualizations. Several empirical studies have compared specific coordinated-visualization interfaces to other approaches such as fish-eye visualizations and detail-only visualizations for browsing hierarchies [CS94] [SSS86] and large 2D spaces [BW90] [PCH92]. In general, these studies indicate an advantage of coordinated visualizations over single detail-only visualizations. However, the studies did not determine why or what aspect of the coordinated visualizations caused improved performance. Was it the additional information displayed in the multiple visualizations or the interactive coordination between them?

Even less is known about users' ability to construct such coordinated exploration environments. Usability work on Apple Dylan [DP95] indicates that once users were shown how to split and link its frames, they were able to remember it. Users were successful with Dylan's single type of data, visualization, and coordination. However, will that carry over to a general coordinated-visualization environment? Can users grasp the notion of establishing different types of coordinations between different types of visualizations? Can users construct appropriate interfaces for themselves this way? Clearly, a deeper level of understanding about users and coordination is needed.

2.5 Summary

These systems provide a foundation of visualization coordination and flexibility that Snap builds on (see Figure 2.14). Snap is a coordination-flexible level system, providing flexibility in data, visualizations, and coordinations. Snap users can construct many common types of coordinations, more than the brushing-and-linking provided by Visage and its cousins. It also uses the object-based approach, which enables more general utility for information visualization interfaces than the attribute-based approach of DEVise. In addition, Snap employs independent visualizations, enabling it to be easily extended by others, whereas each of these systems is monolithic.

	Independent visualizations	Integrated visualizations
Data Flexible	COM, CORBA	Toolkits, C++
Visualization Flexible	Snap	EDA systems, Visage
Coordination Flexible	Snap	DEVise (attribute-based)

Figure 2.14: Constructing coordinated visualizations

Chapter 3: Model of Visualization Coordination

3.1 Background

Snap-Together Visualization is based on a strong underlying model of visualization coordination (the *Snap model*). The goal of this model is to provide a sound theoretical foundation on which the Snap system, user interface, and software architecture can operate. It must have sufficient generality to support:

- common types of information, such as numeric, textual, hierarchical, etc.
- common types of visualizations from the field.
- common types of coordinations for information exploration.

The model must also maintain sufficient simplicity to remain in harmony with the practical architectural goals of integrating independent visualizations. The Snap model formally defines a visualization, coordination, and a coordinated-visualization interface.

In the search to develop this model, several attempted models of coordination were explored but discarded due to their inability to provide a generalizable solution. These included:

- Filter model: a network of filters between visualizations.
- Widget model: user-interface widgets linked using mathematical functions, like constraints.
- User input model: mouse clicks in one visualization are translated to clicks in another visualization.

Two critical realizations led to the development of the current model: First, the recognition of coordination as a visualization problem. That is, coordination deals with each visualization as a whole, not just individual widgets or components within visualizations. Since a visualization is a view of data, it is essentially the data that is being coordinated. Hence, coordination is data dependent.

Second, the recognition of the need for a strong underlying data model to enable a strong coordination model. For example, Isakowitz was successful with RMM [ISB95] because he used a strong underlying relational data model to drive the construction of a web site's pages and hyperlinks.

3.2 Relational Model of Visualization Coordination

The Snap model is based on the relational data model. The relational data model provides several benefits:

- A popular, well-defined, and general-purpose data format.
- Consistency with common visualization practice.
- Unique identifiers (primary-key values) for tuples.
- Well-defined data extraction capability (queries).
- Explicit representation of relationships (joins).

3.2.1 Relational Schemata

With the Snap model, coordinated-visualization interfaces can be constructed to explore relational data. The data is composed of a set of relations, each of which contains a set of tuples. Each relation specifies a list of attributes for which its tuples contain values. Each relation has a primary-key attribute, whose values uniquely identify each tuple in the relation. Relations may also have a set of foreign-key attributes, each of which relates tuples in its relation to tuples in another relation via joins.

Actually, the pure relational data model does not explicitly code the primary-key and foreign-key join relationships between relations. The relationships are only evident when join queries are defined. However, modern relational database management systems such Microsoft Access and Oracle do explicitly specify the relationships in schema diagrams and store them in the form of constraints. A schema diagram shows the relations and their attributes as nodes, and the join relationships as edges between them. For example, Figure 3.1 shows the Access schema diagram of the file-folders database from the example in Chapter 1. There is a one-to-many relationship from folders to files.

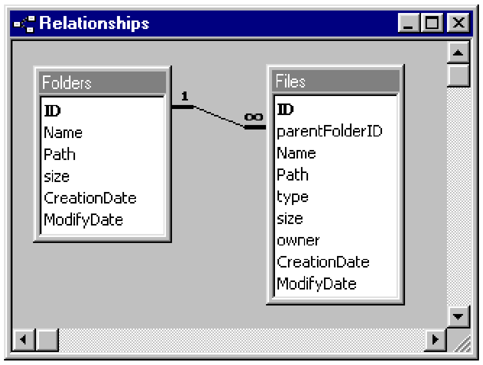


Figure 3.1: Schema diagram

3.2.2 Snap Model Overview

With the Snap model, coordinated-visualization interfaces for relational data are constructed based on the data schema. There is a direct correspondence between concepts in the relational data model and concepts in coordinated-visualization user interfaces (see also Figure 3.2):

Relational Data Model		Coordinated-Visualization User Interface
Relation	=	Visualization
Tuple	=	Item in a visualization
Primary key	=	Item ID
Join	=	Coordination

In Snap, a visualization displays a relation. Coordination between two visualizations is based on the join relationship between their relations. This is somewhat similar to RMM [ISB95], which generates hyperlinks based on join relationships.

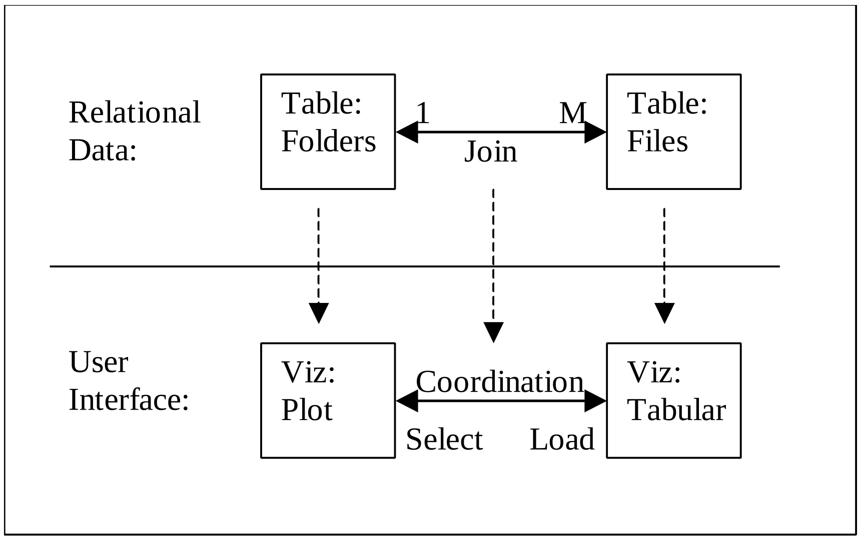


Figure 3.2: Snap model

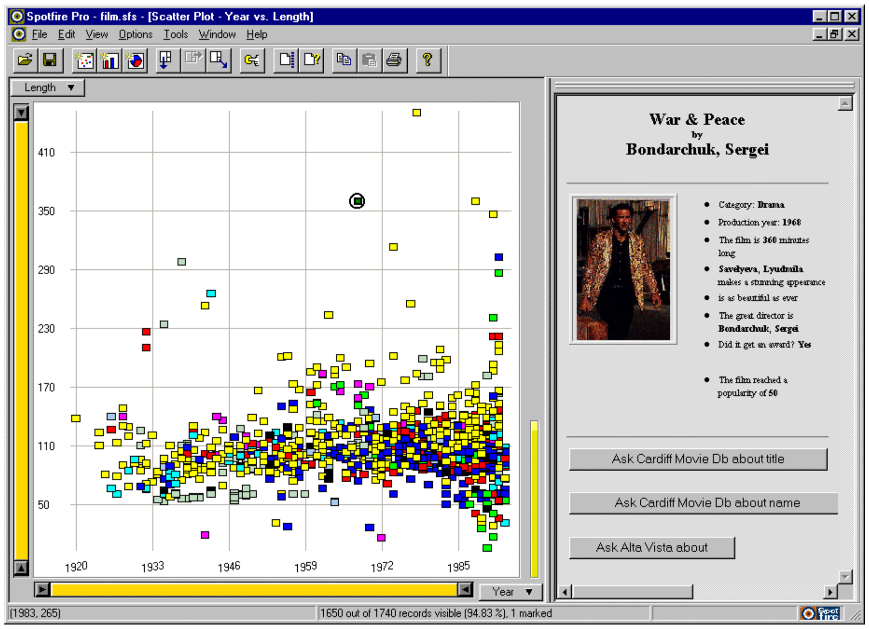
3.2.3 Relations into Visualizations

In the Snap model, a visualization is defined as the use of a visualization type (e.g. scatter plot, Treemap, etc.) to display a single relation from the data. Hence, a visualization is defined as the pair:

Visualization = (visualizationType, relation)

There are two classes of visualizations:

- **Relation visualizations:** In the common case, a relation of many tuples is displayed in the visualization. Generally, each tuple in the relation is depicted as an individual item in the visualization. For example, a scatter plot displays each tuple as a dot using two of its attributes as the coordinates (Figure 3.3). A tabular visualization displays each tuple as a row. The relation must have a primary-key attribute to uniquely identify individual tuples.
- **Single-tuple visualizations:** A single tuple is displayed in the visualization. This type of visualization is often used in two common situations: First, a textual visualization used as the detail view in a details-on-demand coordination to display all the attributes of a single tuple selected in another graphical visualization (Figure 3.3). Second, one or more of the tuple's attribute values are used to locate and display information stored external to the data. For example, a web browser displays a web page given its URL, or a file viewer displays a file given a pathname. These are output-only visualizations.


Spotfire Pro - film.sfs - [Scatter Plot - Year vs. Length]
File Edit View Options Tools Window Help
Length
410
350
290
230
170
110
50
1920
1933
1946
1959
1972
1985
Year
Ask Cardiff Movie Db about title
Ask Cardiff Movie Db about name
Ask Alta Vista about

War & Peace
by
Bondarchuk, Sergei

- Category: **Drama**
- Production year: **1968**
- The film is **360** minutes long
- **Savelyeva, Lyudmila** makes a stunning appearance
- is as beautiful as ever
- The great director is **Bondarchuk, Sergei**
- Did it get an award? **Yes**
- The film reached a popularity of **50**

(1983, 265) 1650 out of 1740 records visible (94.83 %), 1 marked

Figure 3.3: Relation visualization (left), single-tuple visualization (right)

3.2.3.1 Visualization Actions

Visualizations are interactive. Each visualization supports a set of actions that can be performed on individual tuples. These actions can be invoked interactively by users, allowing them to indicate interest in a tuple, or programmatically by the system. These actions are called *primary-key actions*, because the tuple acted on can be identified by its primary-key (PK) value. Example actions include:

- **Select**: select a tuple to visually highlight it. For example, clicking on a dot in a scatter plot colors the dot bright yellow.
- **Scroll**, **zoom**, etc: navigating to a tuple to bring it to the center of view. For example, scrolling a textual list to bring an item to the top of the window, or zooming onto a node in a Treemap, or centering the focus on an item in a fish-eye visualization.

Since actions are visualization dependent, each visualization defines the set of actions it supports (providing a name for each action) according to the interaction mechanics of the visualization's user interface. In general, visualizations have two types of actions: selection actions and navigation actions. For example, Treemap has three actions:

- Select click: clicking on a node highlights it with a yellow rectangle.
- Select mouse-over: moving the mouse over a node highlights it with a white rectangle.
- Zoom: double-clicking a node zooms that node to fill the view.

In addition, each visualization also has a **load** action. The load action loads and displays only the specified tuple(s) from the relation into the visualization. When used as a primary-key action, a single tuple identified by its primary-key value is loaded into the visualization. This is used with single-tuple visualizations.

The load action can also be used as a *foreign-key action*. In this case, multiple tuples, identified by the value of one of their foreign-key (FK) attributes, are loaded into the visualization. The specified value is thus a primary-key value of a tuple in a joined relation, and hence the loaded tuples are all related to that tuple.

When a load action is invoked, the visualization is first cleared so that only the tuples from the current load action are displayed. This enables a visualization to be used to display different portions of a large relation based on external input. If the load action is not used, then the entire relation is displayed in the visualization.

Hence, an action invocation can be expressed as a triple:

Invocation = (visualization, action, PKvalue)

That is, *action* is invoked in *visualization* on a tuple identified by *PKvalue*. If the action is *load*, then it must also specify primary-key or foreign-key action. Foreign-key actions specify the foreign-key attribute to use. E.g. load(PK), or load(FK_i).

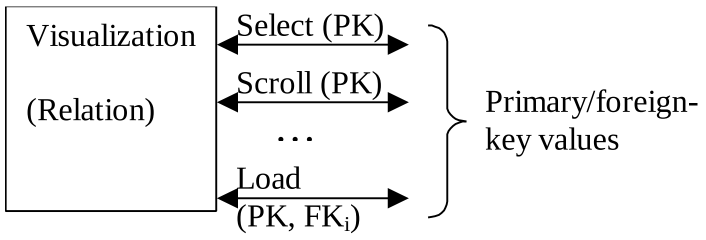


Figure 3.4: Diagram of a visualization and its actions

3.2.4 Coordinating Visualizations

In the Snap model, a coordination tightly couples an action in one visualization to an action in another visualization. Thus, when users invoke the former action, Snap automatically invokes the latter, and vice versa. The tuples acted on in each visualization are related by the join between their relations. When users invoke one of the actions, joining the visualizations' relations determines the corresponding tuples to act on in the other visualization.

Hence, a coordination is defined as an action-invocation pair:

$$\text{Coordination} = ((\text{visualization}_1, \text{action}_1, \text{PKvalue}), (\text{visualization}_2, \text{action}_2, \text{PKvalue}))$$

The PKvalue is bound between the two invocations. Since this can be assumed, the short-hand notation is:

$$\text{Coordination} = ((\text{visualization}_1, \text{action}_1), (\text{visualization}_2, \text{action}_2))$$

A coordination between a pair of visualizations is established by choosing the actions to tightly couple. The join relationship between the visualizations' relations determines which of the three possible combinations of primary-key and foreign-key actions can be used:

3.2.4.1 One-to-One: Primary-Key to Primary-Key

This is a primary-key to primary-key relationship, and is often the result of displaying different projections of the same table in multiple visualizations. A primary-key action in one visualization can be tightly coupled to a primary-key action in the other, linking their primary-key values. Hence, when one of the actions is invoked, the other is also invoked on the same primary-key value.

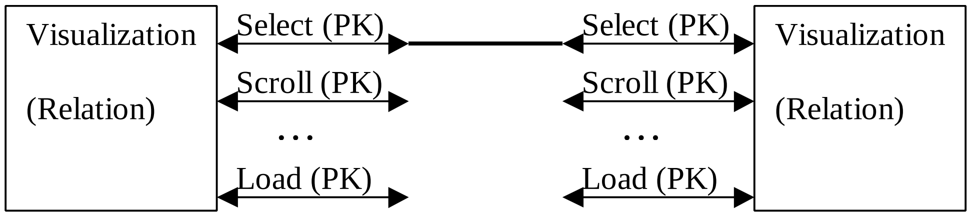


Figure 3.5: One-to-one coordination, e.g. brushing and linking

Examples of one-to-one type coordinations, from Chapter 1, are:

- **Brushing and linking**:

  Coordination: ((visualization$_1$, *select*), (visualization$_2$, *select*))

  Operation: Selecting an item in one visualization also selects (highlights) the corresponding item in the other visualization. For example, in the file-folder example in Chapter 1, selecting a folder in the Hyperbolic Tree highlights that folder in the scatter plot. Figure 3.5 shows the coordination specification.

- **Overview and detail**:

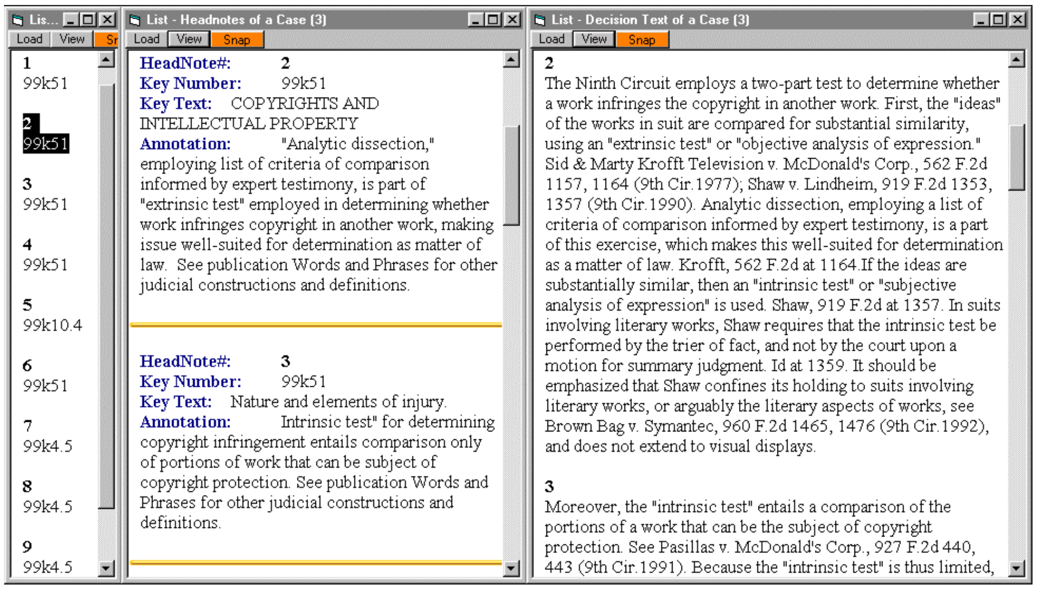
  Coordination: ((overview, *select*), (detail, *scroll*))

  Operation: Selecting an item in the overview scrolls (or more generally navigates) the detail visualization to the details of that item. Likewise, scrolling the detail selects the currently viewed item in the overview. For example, in Figure 3.6, selecting a document section from the list on the left jumps the scrolling document text on the right to that section.

- **Synchronized scrolling**:

  Coordination: ((visualization$_1$, *scroll*), (visualization$_2$, *scroll*))

  Operation: Scrolling through a list of tuples in one visualization also scrolls to corresponding items in another visualization. For example, in Figure 3.6, scrolling the document text on the right also scrolls the document annotations in the center to the corresponding section.


Lis...
Load
View
Sr
List - Headnotes of a Case (3)
Load
View
Snap
List - Decision Text of a Case (3)
Load
View
Snap

1
99k51

2
99k51

3
99k51

4
99k51

5
99k10.4

6
99k51

7
99k4.5

8
99k4.5

9
99k4.5

HeadNote#: 2
Key Number: 99k51
Key Text: COPYRIGHTS AND INTELLECTUAL PROPERTY
Annotation: "Analytic dissection," employing list of criteria of comparison informed by expert testimony, is part of "extrinsic test" employed in determining whether work infringes copyright in another work, making issue well-suited for determination as matter of law. See publication Words and Phrases for other judicial constructions and definitions.

HeadNote#: 3
Key Number: 99k51
Key Text: Nature and elements of injury.
Annotation: Intrinsic test" for determining copyright infringement entails comparison only of portions of work that can be subject of copyright protection. See publication Words and Phrases for other judicial constructions and definitions.

2
The Ninth Circuit employs a two-part test to determine whether a work infringes the copyright in another work. First, the "ideas" of the works in suit are compared for substantial similarity, using an "extrinsic test" or "objective analysis of expression." Sid & Marty Krofft Television v. McDonald's Corp., 562 F.2d 1157, 1164 (9th Cir.1977); Shaw v. Lindheim, 919 F.2d 1353, 1357 (9th Cir.1990). Analytic dissection, employing a list of criteria of comparison informed by expert testimony, is a part of this exercise, which makes this well-suited for determination as a matter of law. Krofft, 562 F.2d at 1164.If the ideas are substantially similar, then an "intrinsic test" or "subjective analysis of expression" is used. Shaw, 919 F.2d at 1357. In suits involving literary works, Shaw requires that the intrinsic test be performed by the trier of fact, and not by the court upon a motion for summary judgment. Id at 1359. It should be emphasized that Shaw confines its holding to suits involving literary works, or arguably the literary aspects of works, see Brown Bag v. Symantec, 960 F.2d 1465, 1476 (9th Cir.1992), and does not extend to visual displays.

3
Moreover, the "intrinsic test" entails a comparison of the portions of a work that can be the subject of copyright protection. See Pasillas v. McDonald's Corp., 927 F.2d 440, 443 (9th Cir.1991). Because the "intrinsic test" is thus limited,

Figure 3.6: Case-law document browser: overview and detail, and synchronized scrolling

3.2.4.2 One-to-Many: Primary-Key to Foreign-Key

A primary-key to foreign-key relationship indicates a hierarchical structure between the relations. Each parent tuple in the first relation has many child tuples in the second relation.

The allowable combination is: tightly couple a primary-key action in the visualization on the *One* side of the relationship with a foreign-key action on the *Many* side. This links the primary-key value of the primary-key action to the foreign-key value of the foreign-key action. When the primary-key action is invoked, the foreign-key action is also invoked using the primary-key value as the foreign-key.

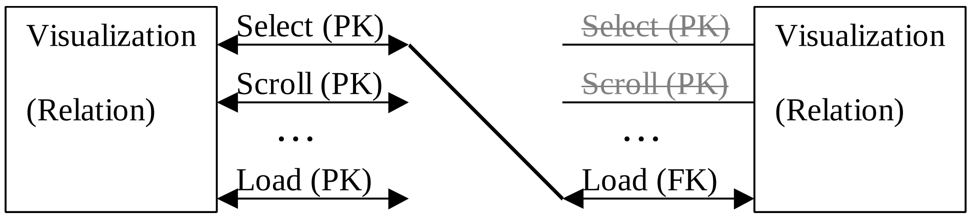


Figure 3.7: One-to-many coordination, e.g. drill down

A common coordination for this relationship type is:

- **Drill down**:

  Coordination: ((parentViz, *select*), (childViz, *load(FK)*))

  Operation: Selecting an item in the parent visualization loads related items into the child visualization. For example, in the file-folder example, selecting a folder in the plot loads and displays the files related to that folder in the tabular visualization.

3.2.4.3 Many-to-One-to-Many: Foreign-Key to Foreign-Key

This is an implicit relationship that occurs when two relations share a common foreign-key attribute. That is, a shared parent relation has a one-to-many join with both relations.

In this case, a foreign-key action can be coupled to a foreign-key action. A *load(FK)* to *load(FK)* coordination is often used when it is desired for two visualizations to display different descendants of the same parent. For example, the U.S. states have counties and voting districts. Two visualizations could be coordinated to always display the counties and districts (respectively) of the same state.

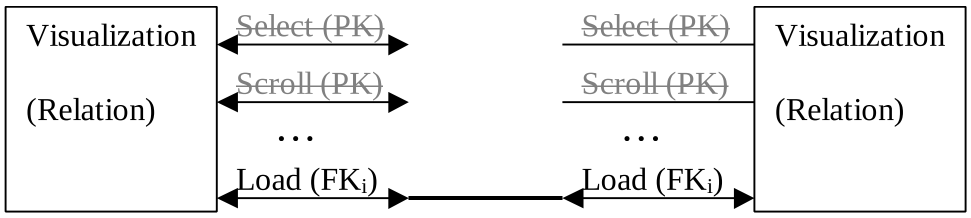


Figure 3.8: Many-to-one-to-many coordination

There is one restriction on foreign-key actions. While a visualization may have several different load actions available (primary key and for each foreign key), only one of these load actions can be tightly coupled at a time. Thus, all other visualizations that coordinate to a visualization's load action must use the same key attribute.

3.2.5 Schema Management

In the Snap model, if additional visualizations or coordinations are desired beyond what is available in the data schema, then additions can be made to the schema. That is, if the schema graph does not translate to the desired coordinated-visualization behavior, than simply modify the schema. Hence, with Snap, advanced coordination is simply a schema manipulation problem rather than a custom user-interface programming problem.

Schema management is used in two situations: creating queries to generate desired visualizations, and establishing relationships to generate desired coordinations.

First, when the data tables in the schema do not provide the appropriate relation needed for a visualization, then a query (view) can be created to generate the desired relation. There are three common situations in which queries are used to generate desired visualizations. Each is based on a single source table. The query can be added to the schema, with a join relationship to its source table. The query must also inherit a primary-key attribute from its source table.

- **Projection**: This is often used when only a subset of the attributes of a relation are needed for a visualization. The query is one-to-one with the original relation.
- **Selection**: This is used to display a subset of the tuples of a relation. It is one-to-one with the original relation.
- **Aggregation**: This aggregates tuples in a relation, and is often used to create drill-down coordinations. It is one-to-many to the original relation. The GROUP-BY attribute is used as its primary-key attribute.

Second, if the schema has no direct primary-key or foreign-key relationship between two relations, then a coordination cannot be established between their visualizations. However, if there is an indirect path through other relations, then it is generally possible to modify the schema to generate the appropriate behavior. For example, a query could be created that joins the relations along the path.

3.3 Graph Model of Composite Coordinations

The previous section examined coordinating two visualizations together. Now, this is expanded to composing many visualizations with many coordinations. Coordinated-visualization interfaces can be defined using a graph model. Expanding on the above definitions for a visualization and a coordination, a coordinated-visualization interface (CVI) is defined as a set of visualizations (V) and a set of coordinations (C) between them:

CVI = (V, C), where

$V = \{v_1, \ldots, v_n\}$, v_i = (visualizationType, relation)

$C = \{c_1, \ldots, c_m\}$, $c_i = ((v_j, \text{action}_j), (v_k, \text{action}_k))$, where $v_j, v_k \in V$.

This is a graph in which nodes are visualizations and edges are coordinations. Edges are labeled at both ends with the actions that are tightly coupled in the coordination. In the Snap model, since visualizations correspond to relations and coordinations correspond to joins, the coordination graph corresponds directly to the data schema graph. For example, Figure 3.9 shows the coordination graph for the file-folders interface for system administrators from Chapter 1.

This model indicates several properties that further describe how coordinated-visualization interfaces operate as follows.

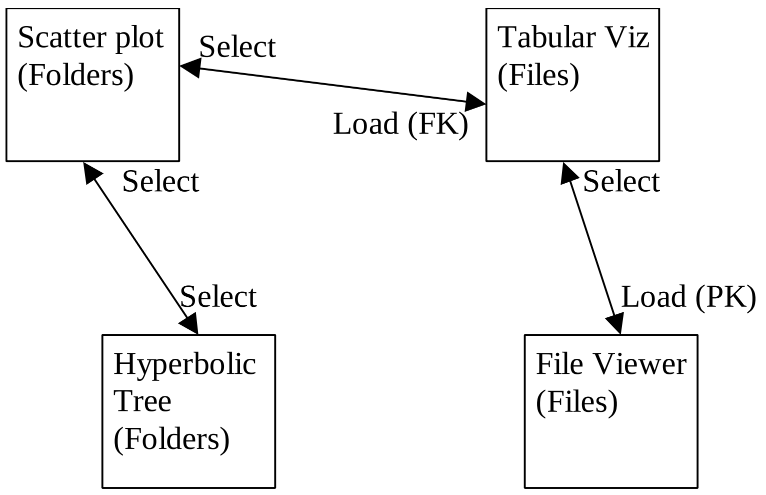


Figure 3.9: Coordination Graph

3.3.1 Commutative

$$((v_j, action_j), (v_k, action_k)) \Leftrightarrow ((v_k, action_k), (v_j, action_j))$$

Snap coordinations are bi-directional, so that either action triggers the other. For example, selecting a folder in the Hyperbolic Tree highlights it in the scatter plot, and by commutativity, selecting in the plot highlights in the Hyperbolic Tree. Note, however, that each action is strictly tied to its visualization. Hence:

$$action_j \neq action_k \Rightarrow ((v_j, action_j), (v_k, action_k)) \neq ((v_k, action_j), (v_j, action_k))$$

3.3.2 Transitive

$$((v_i, action_i), (v_j, action_j)) \wedge ((v_j, action_j), (v_k, action_k))$$
$$\Rightarrow ((v_i, action_i), (v_k, action_k))$$

Coordinations can be chained end-to-end. Invoking an action at one end will propagate down the chain, triggering actions at each visualization. All visualizations related by transitivity to the visualization where the action is invoked will have their coordinated actions invoked. For example, brushing-and-linking can be established across three visualizations. In the file-folder example, selecting a folder in the

Hyperbolic Tree also selects it in the plot, which in turn loads files into the tabular visualization.

Invoking an action on any visualization in the coordination graph essentially initiates a graph traversal. Coordinations only propagate at each visualization if the incoming action from one coordination matches the out-going action of the next. For example, selecting a folder in the plot loads files into the tabular visualization, but does not cause any action on the file viewer. Formally:

$$((v_i, action_i), (v_j, action_j)) \wedge ((v_j, action_x), (v_k, action_k))$$
$$\not\Rightarrow ((v_i, action_i), (v_k, action_k))$$

Hence, a *connected component* can be defined as a subset of a CVI containing all visualizations and coordinations related by transitivity to a single visualization action invocation. Connected components are essentially spheres of interaction in the coordinated-visualization interface. The file-folder example has two connected components: selecting folders, and selecting files (Figure 3.10).

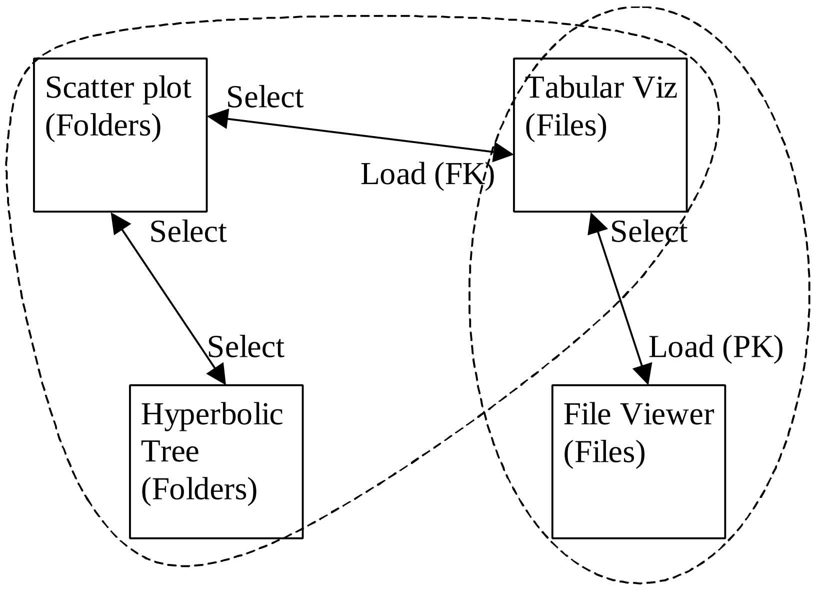


Figure 3.10: Connected components

Using the expanded notation for coordination (viz, action, PKvalue) with transitivity shows that only the primary-key value of the action initiated by the user is propagated during the traversal of a connected component.

The transitive property enables the deduction of coordinations. For example, given three visualizations and two transitive coordinations connecting them, the third coordination can always be deduced. Figure 3.11 shows the 4 possible transitive combinations of the three relationship types (PK-PK, PK-FK, FK-FK). For example, in the top right diagram describes the file-folder example as C=Hyberbolic, B=plot, A=tabular. The coordination C-A can be derived as *select* to *load*.

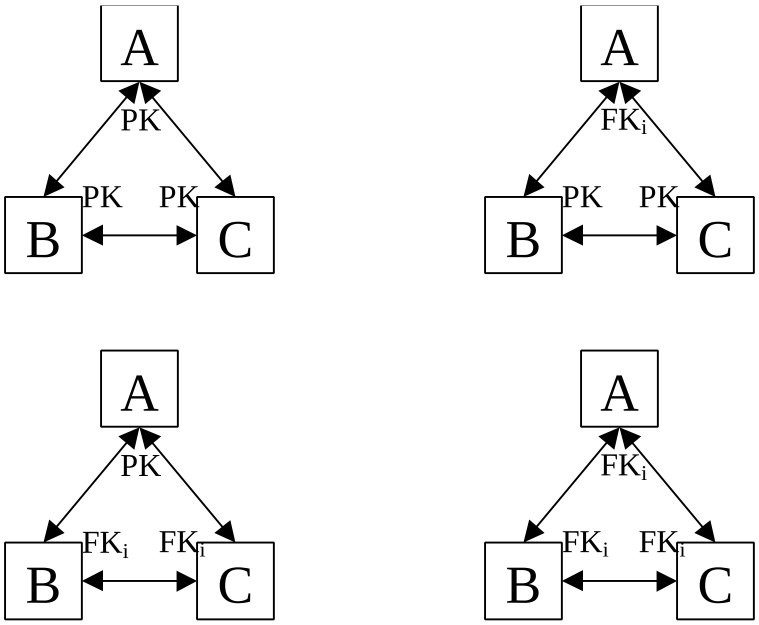


Figure 3.11: Deriving coordinations with transitivity

3.3.3 Conflict Free

An important property of the Snap model is that its coordinations are conflict free. In designing systems that use coordination, there is often a concern about possible conflicts due to cycles in the coordination graph. For example, in an exotic scenario, selecting a U.S. state on a map might coordinate to highlight the state's governor in a list of the 50 governors, which might then coordinate to highlight the governor's birth state on the map. This creates a cycle and would then attempt to highlight the governor of that state, etc. A conflict occurs when the same action is invoked on the same visualization twice (or more) with different primary-key values in a single coordination execution cycle. Conflicts can result in endless looping or mismatched state between visualizations.

However, cycles in Snap are always redundant and never conflicting. That is, when a coordination propagation visits a visualization a second time due to a cycle, the primary-key value given is always the same value for both visits, resulting in a redundant action and not a conflict. This is because only a single primary-key value is propagated during a single coordination traversal. This is easily proven using the transitive property. When deriving a self-coordination using transitivity around a cycle, the result clearly shows the equality of the PKvalue:

$$((v_j, \text{action}_j, \text{PKvalue}), (v_j, \text{action}_j, \text{PKvalue}))$$

Hence, the Snap model would not allow the above exotic example to be constructed. The offending component is the 'birth state' coordination. This is a one-to-many relationship from states to governors, and hence cannot support a *select* (PK) to *select*

(PK) coupling. A potential solution is to employ a third visualization to display the birth state. From a user's point of view, this would make more sense anyway.

A mark-on-visit traversal algorithm can be used to detect and stop cycles (self-coordinations). Each action of a visualization is marked independently. Thus, a visualization can be visited twice, but only for different actions.

3.3.4 Subgraphs

Subgraphs can be easily added to or extracted from a coordinated-visualization interface. These subgraphs are themselves CVI's. Hence, this enables the saving and reusing of CVI's. A saved coordinated-visualization interface can be immediately instantiated and coordinated to other visualizations. Essentially, this enables the construction of composites as new primitives. This powerful notion resembles how programs or macros are saved for later use in more complex programs.

3.4 Applications and Limitations

The Snap model captures a variety of types of data, visualizations, and coordinations that are commonly used in information visualization. Appendix A describes Snap's use in a diverse set of examples, including census statistics, GIS maps, case-law textual information, photo libraries, hierarchical file structures, web sites, and address databases. The model exploits existing functionality of visualizations and exploits data relationships to enable coordinations for navigating and exploring information.

The Snap model focuses on interaction with individual data objects (i.e. relational tuples). Hence, this model is not well suited for attribute-based spatial coordinations, such as large 2D or 3D image browsing applications and scientific visualization.

Navigation in these applications is often based on pixels or voxels, as in the Visible Human Explorer [NSP96] for example. While it would be possible to model images in a relational data model (e.g. pixel = tuple), it simply is not very practical.

However, such spaces often have meaningful objects embedded, in which case the Snap model is very appropriate. For example, the Visible Human 3D image data contains segmented anatomical structures with links to databases of anatomical information such medical terminology dictionaries. Figure 3.12 shows a mockup from early work on Snap which uses brushing and linking between anatomical objects in the image data (like an image map) and terms in the Medical Subject Headings hierarchical dictionary [Nor98]. In fact, based on extensive work on 3D image browsing with medical domain experts [CSP97], Konstan discovered that the experts desired these types of cross-media database coordinations more than the spatial navigation coordinations [Kon97].

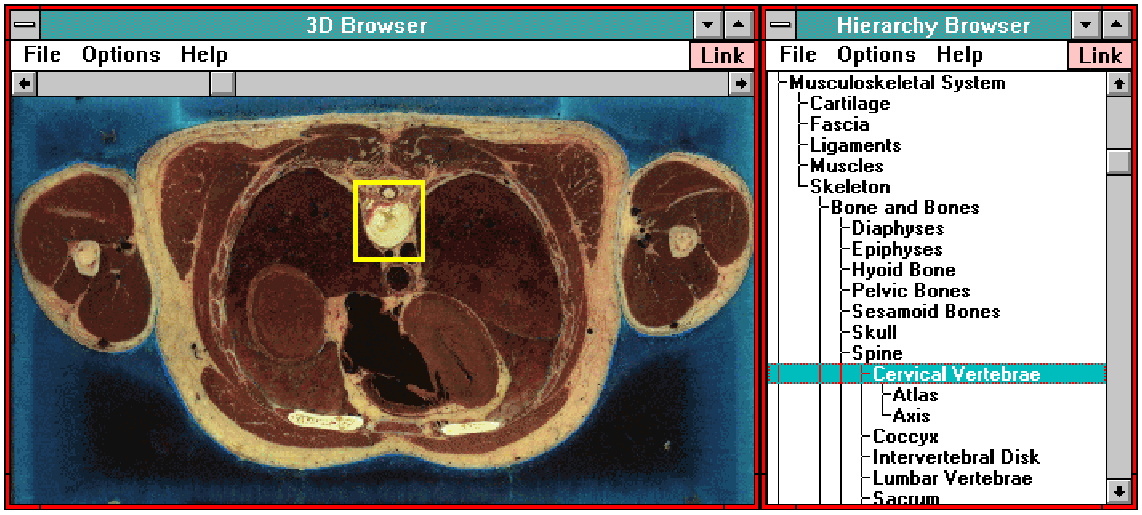


Figure 3.12: Relationships between image and textual data

The embedding of meaningful objects is also found in other domains as well. For example in continued work on Pad++ [BH94], a zoomable user interface in which 2D spatial navigation is primary, Bederson has increasingly employed an object-based approach where users click hyperlinks to navigate between objects rather than manual spatial navigation.

Dynamic Queries [AS94] and coordinating plot axes (as in DEVise [LRB97]) are also attribute based. Although these could be handled in Snap by enumerating matched objects, this is not very efficient. Dynamic Queries requires specialized data structures and algorithms in each visualization, so is inherently in conflict with the goal of using independent visualizations anyway. It would be interesting to explore how the attribute-based approach could be combined with the Snap model.

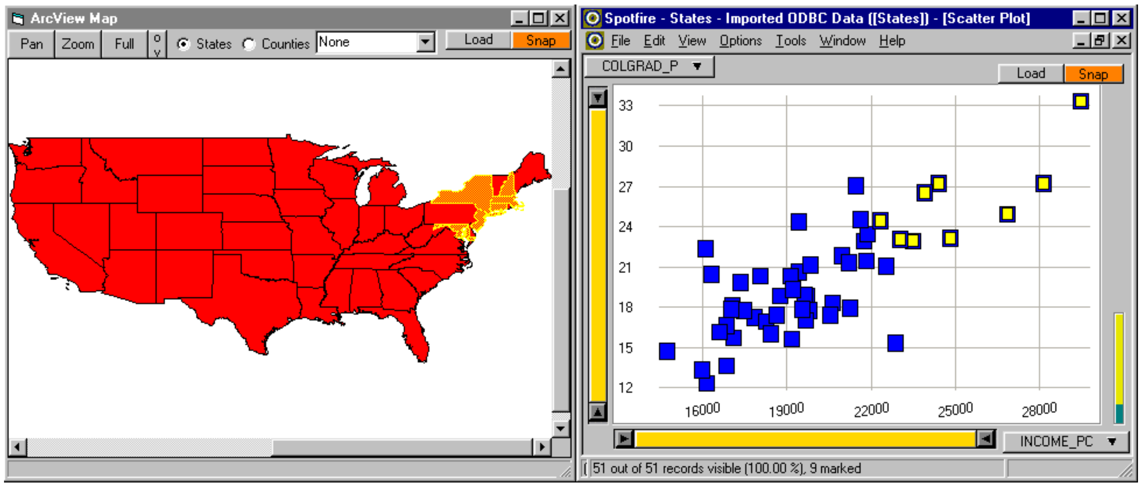
3.5 Extensions to the Model

Snap's conceptual model is intentionally designed with simplicity to simultaneously capture the need for a model of visualization coordination as well as meet the practical architectural goals (as discussed in Chapter 5). However, this model may be extended in several ways:

3.5.1 Multiple-Tuple Actions

Some actions in some visualizations may be able to act on multiple tuples. Instead of acting on a single primary-key value, these actions could act on a set of values. For example, multiple selection [Wil96] is often used for brushing-and-linking coordinations to enable users to highlight several tuples simultaneously. Clearly, this does not apply to all actions. Some actions, such as scroll, are semantically single-tuple. Others are limited by the visualization's software architecture, e.g. Treemaps can only select one node. However, visualizations could mark their actions as single- or multiple-tuple capable. Then, Snap could allow actions of the same cardinality to be tightly coupled in coordinations.

In fact, in continued work on Snap at the Census Bureau, multiple selection has been added. For example, in Figure 3.13, selecting the high income and highly educated U.S. states in the scatter plot (using Spotfire's lasso selection capability) reveals that those states are all in the northeast, the DC to Boston corridor.



51 out of 51 records visible (100.00 %), 9 marked

Figure 3.13: Brushing and linking with multiple-tuple selection

3.5.2 Unions and Intersections

The drill-down coordination enables users to select a parent tuple in one visualization to load and display its children tuples in another (1-M). This could be extended for union and intersection by using multiple selection and allowing simultaneous use of different foreign-key load actions as follows:

- **Union**: Selecting multiple parent tuples in the same parent visualization would display the union of their children. For example, selecting two folders would display the files of both in the tabular visualization.
- **Intersection**: Selecting multiple parent tuples from different parent visualizations would display the intersection of their children. For example, selecting a folder and a user would display only the files in that folder owned by that user. This would allow the construction of simultaneous-menu applications [HKV00]. This requires allowing load actions on different foreign keys simultaneously.

This approach would enable users to select from several different overviews to filter items in a main visualization, enabling functionality very similar to Dynamic Queries.

3.5.3 Other Foreign-Key Actions

It would also be possible to enable multiple-tuple actions to act as foreign-key actions. For example, selecting a parent tuple in one visualization might coordinate across a 1-M join to select and highlight all its children tuples in another visualization. This could be used for brushing-and-linking across many-to-many relationships.

However, this will likely introduce confusion for users. Primary-key actions and foreign-key actions are semantically different, but users would not be able to distinguish primary-key selections from foreign-key selections in the visualization. For example, what should happen if users select a child tuple in the latter visualization? Also, should the foreign-key selection of children tuples initiate new primary-key selection actions for each? This modification would introduce the potential for conflicts in coordination.

3.6 Summary

The Snap model provides a solid, well-founded basis for visualization coordination. It is based on the relational data model. Visualizations display relations, and coordinations correspond to one-to-one and one-to-many join relationships. A graph model describes the coordination of multiple visualizations. The Snap model is the underlying basis for the Snap user interface and software architecture.

Chapter 4: User Interface for Coordination Construction and Operation

4.1 Background

The Snap-Together Visualization user interface enables data users to quickly and dynamically construct coordinated-visualization interfaces without programming. Then, they can efficiently explore their data using these powerful coordinated-visualization interfaces that are custom tailored to their data, tasks, and preferences.

Snap is used in two modes. Users first construct interfaces, then operate them to explore. However, there is not a distinct mode switch between modes. Users can interchange activities on the fly as needed.

Chapter 1 provided an overview scenario of the Snap user interface. This chapter proceeds to describe the interface in detail.

4.1.1 Users

As indicated by the study in Chapter 6, the users that construct interfaces with Snap are likely to be the more data-savvy users or data owners, such as analysts or data providers. These highly motivated users are familiar with the general content and structure of the data (e.g. the data schema), and have accumulated some experience in constructing with Snap.

Some of these users construct interfaces for their own use. For example, an analyst at the Census Bureau might quickly snap together an interface while examining trends in newly collected survey results. The analyst could also present findings to co-workers using the same interface.

Users can also construct interfaces for use by others. For example, a data-product specialist at the Census Bureau could construct an interface to accompany the distribution or publication of census-2000 population statistics. Then, casual readers such as policy makers or business owners could easily examine the data, using the pre-constructed interface, and make decisions.

As an in-between case, an analyst might construct interfaces for use by other analysts in the organization, similar to the way they share Excel macros [NM91].

Several enhancements to the Snap interface are described later in this chapter that are aimed at enabling even casual end users to construct coordinated-visualization interfaces themselves. This is accomplished by using direct manipulation techniques to reduce users' learning time, performance time, and error rates.

4.1.2 Requirements

The Snap user interface is soundly based on the underlying Snap model for visualization coordination. For construction, the model completely specifies what choices users make to specify a new coordinated-visualization interface (CVI). Recalling the definition:

$$\text{CVI} = (V, C), \text{ where}$$

$$V = \{v_1, \ldots, v_n\}, \quad v_i = (\text{visualizationType}, \text{relation})$$

$$C = \{c_1, \ldots, c_m\}, \quad c_i = ((v_j, \text{action}_j), (v_k, \text{action}_k)), \text{ where } v_j, v_k \in V.$$

Hence, to construct a coordinated-visualization interface, users must:

1. create visualizations by matching relations to visualization types, and

2. coordinate visualizations by selecting pairs of visualizations and specifying actions to tightly couple in each.

In addition, the possible choices of actions to tightly couple are specified by the data schema, the one-to-one and one-to-many relationships. These definitions provide users with syntactic guidance only. Semantic guidance comes from the semantics of the data schema and the desired tasks to support.

4.2 Coordination Construction

Snap's user interface employs a two-step approach to construction. Users first open and display relations in visualizations. Then, they coordinate the visualizations by tightly coupling actions between the visualizations.

Implementation note: Snap supports database formats that have ODBC drivers, such as Microsoft Access or Oracle (see Implementation section of Chapter 5). To edit the database and schema, the database's native software is used. For example, Access databases are manipulated using Access's GUI. For databases that do not have the necessary software, Snap provides a simple SQL text editor to add and edit queries. The description in the remainder of this chapter assumes an Access database.

4.2.1 Relations into Visualizations

Starting Snap displays the Snap Menu window. To begin construction and exploring a database, users first open the database using Snap. Any database (of the supported formats) can be opened with Snap. That is, Snap is not hard wired to a specific database or schema. Snap determines the schema from the database.

The Snap Menu (Figure 4.1) displays a list of the tables and queries in the database (left). It also displays a menu of the available visualization types (right). To display a relation in a visualization, users simply drag the desired table or query name onto a visualization-type button (or select a relation and click a visualization button). The visualization tool opens and the relation is loaded and displayed. Users can open as many as needed.

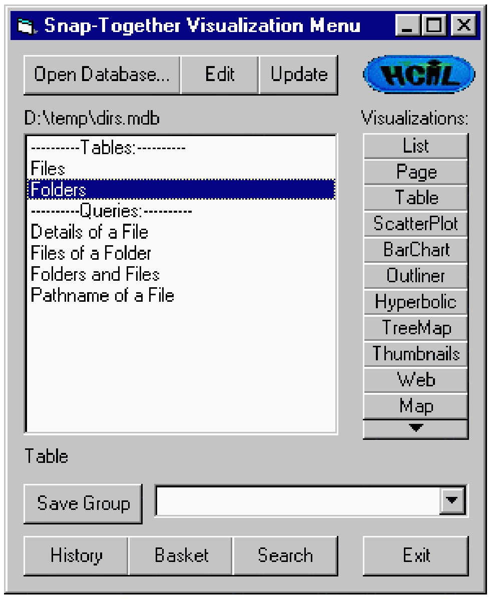


Figure 4.1: Snap Menu

4.2.1.1 Visualization Types

The current implementation has the following visualization types:

- **Scrolling list:** Displays each tuple like a textual report with each attribute on a new line (Figure 4.2). Particularly useful for long text (e.g. memo fields). Multiple tuples are separated by a horizontal rule. Actions: *select* a tuple, and *scroll* to a tuple.
- **Paging list:** Like scrolling list, but displays only one tuple at a time. A paging bar enables navigation to the other tuples. Actions: *select* a tuple, and *page* to a tuple.

- **Table:** Standard rows (tuples) and columns (attributes) display. Actions: *select* a tuple, *scroll* a tuple to the top.
- **Spotfire:** Commercial dynamic-query software, including scatter plot, bar chart, pie chart. Actions: *select* a tuple by click or mouse-over.
- **Outliner:** Standard nested-indented hierarchy widget. Actions: *select* a tuple.
- **Treemap:** Research software that displays hierarchies by area-coded slice-and-dice containment. Actions: *select* a tuple (node) by click or mouse-over, *zoom* onto a tuple.
- **Hyperbolic Tree:** Commercial Java applet (Inxight Software) that displays hierarchies as a radial fish-eye. Actions: *select* and center focus on a tuple.
- **Internet Explorer (IE):** Accepts a relation with a single tuple and a single attribute which contains the URL or pathname of the web page, folder, or file (file viewer) to display. Actions: none, output only.
- **Image Thumbnails:** Displays a set of thumbnail images in a flexible manner, using pathnames attribute. Actions: *select* a tuple (image), *zoom* a tuple.
- **Image Maps:** Uses IE to display an image map. Actions: *select* a tuple (region of the image map).
- **ArcView Maps:** Commercial GIS software that displays choropleth (colored by attribute) geographic maps. Actions: *select* a tuple (geographic entity), *zoom* on a tuple.

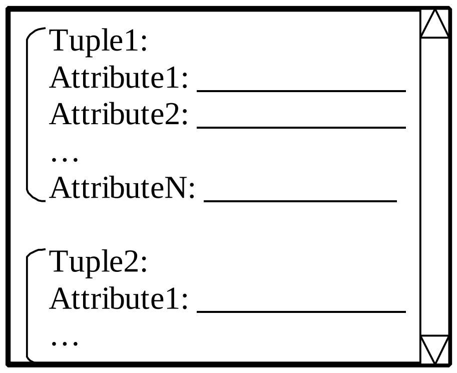

Tuple1:
Attribute1: ______________
Attribute2: ______________
…
AttributeN: _____________
Tuple2:
Attribute1: ______________
…

Figure 4.2: Scrolling list visualization

The hierarchical visualizations require a pathname attribute that specifies the hierarchical structure of the tuples. There are two variations:

- *Complete* hierarchies have a tuple for each node in the hierarchy. For example, in the file-folder example, each folder in the hierarchy is represented by a tuple.
- *Leaf-only* hierarchies have tuples for only the leaf nodes. For example, a relation of U.S. states might organize the states into six major regions. Hence, the region level of the hierarchy does not have tuples, only the states at the leaf level do.

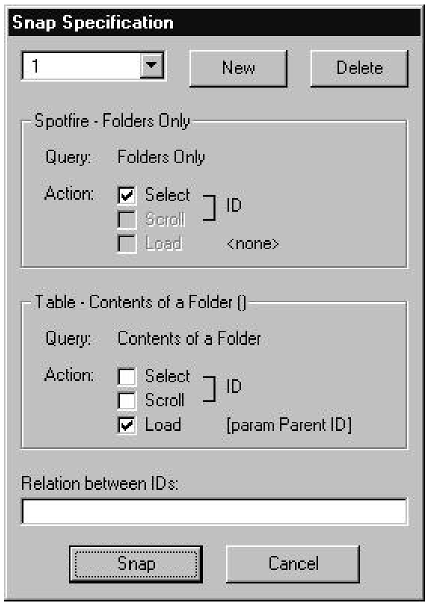
4.2.2 Coordinating Visualizations

When opening a visualization tool, Snap automatically adds a “snap” button Snap to its window in the upper right corner. This is intended to be similar to the way the window manager adds minimize, maximize and close buttons to each window.

To establish a coordination between two visualizations (“snap them together”), users first identify the pair by dragging the snap button from one of the visualizations to the snap button of the other visualization (as shown in the Chapter 1 scenario). This drag-and-drop approach for selecting pairs of visualizations is similar to that of

LinkWinds [JBO94] and Apple Dylan [DP95], although the latter distinguishes between output and input buttons.

Then, the Snap Specification dialog is displayed (Figure 4.3). The Snap Specification has two group boxes. The top box displays information about the first visualization (at the source of the drag-and-drop), and the bottom box displays information about the second visualization (the destination of the drop). The order of the visualizations is not important, since coordinations are bi-directional.



Spotfire - Folders Only

Query: Folders Only

Action: [✓ Select] ID
Scroll
Load <none>

Table - Contents of a Folder ()

Query: Contents of a Folder

Action: Select ID
Scroll
[✓ Load] [param Parent ID]

Relation between IDs:

Figure 4.3: Snap Specification dialog

The information displayed about each visualization includes: the title of the visualization, the name of the table or query in the visualization, the set of actions available for tight-coupling, and the primary-key and foreign-key attribute names. There are three action slots for each visualization. The actions shown in these slots are completely determined by the visualization. If the visualization offers fewer than three actions, then the remaining slots are grayed out.

Users can then select which of the actions to tightly couple. For example, choosing the *select* actions of both visualizations will create a brushing-and-linking style coordination.

The current implementation does not have access to information about the relational joins in the data schema. Hence, users must enforce the primary-key action and foreign-key action combination rules themselves. The display of the key attribute names helps users remember the join relationships. The system does attempt to guess the join by matching the key names, but is fallible of course.

To use the *load* action, users first create a parameterized selection query that extracts tuples by matching the appropriate primary-key or foreign-key attribute to the given key value. Then, they open this query in a visualization. The presence of the parameterized query enables the visualization's load action. They can then tightly couple the load action in a coordination. The parameterized query enforces the rule that only one foreign-key or primary-key attribute can be used in a visualization's load action at a time.

For example, in the file-folders example in Chapter 1, the *select* action of the scatter plot is coordinated to the *load* action of the tabular visualization. The tabular visualization displays the results of a query like:

```
SELECT * FROM files WHERE files.parentFolderID = <parameter>
```

4.2.2.1 Modifying Coordinations

Users can edit coordinations by clicking on a visualization's snap button. The Snap Specification dialog displays coordinations to that visualization. If there are multiple coordinations, the combo box at the top of the dialog is used to flip through them. Users can then change the choice of tightly coupled actions or delete a coordination entirely.

4.2.2.2 Coordination Suggestion

When the system can determine the join relationship between the visualizations, it automatically suggests the following common coordinations in the Snap Specification dialog:

- Primary key to primary key: suggests *select* to *select*, for brushing-and-linking.
- Primary key to foreign key: suggests *select* to *load*, for drill-down.
- Foreign key to foreign key: suggests *load* to *load*.

Users can immediately accept the suggestion, or override with their own choices.

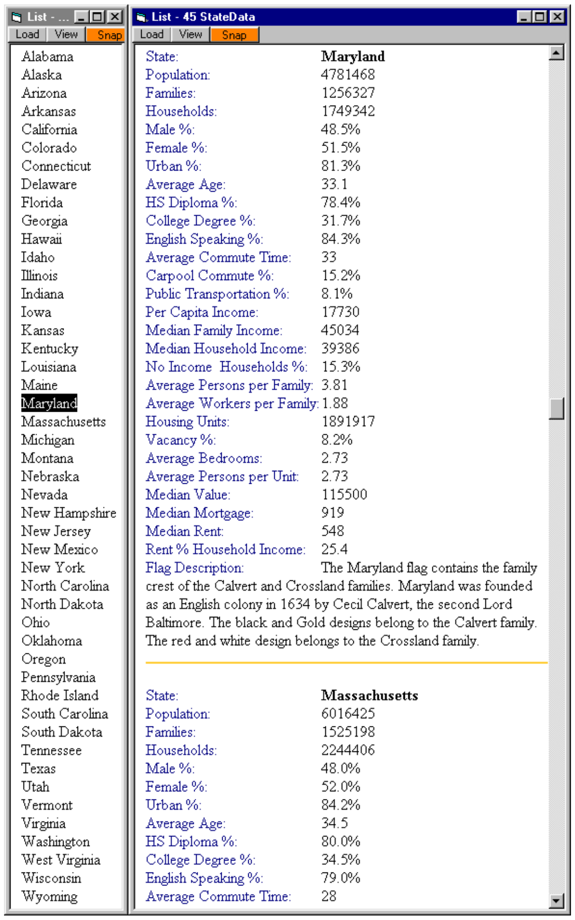
4.3 Coordination Operation

Once the coordination has been established, users can then operate the now coordinated visualizations. These coordinated-visualization interfaces significantly improve users' performance in many tasks as shown in the user studies in Chapter 6. Users are better able to explore, understand, and discover new information.

In addition to guiding the construction of coordinations, Snap's model also specifies how the coordinations operate once constructed. The commutative and transitive properties of the graph model lead to bi-directionality and propagation in the user interface.

4.3.1 Bi-Directionality

When users perform either of the actions tightly coupled in a coordination, the other is also executed. For example, Figure 4.4 shows an interface constructed with Snap for browsing information on the U.S. states using an overview-and-detail coordination. Selecting a state in the overview immediately scrolls the detail to the information about that state. Likewise, scrolling through the detail highlights the name of the currently viewed state in the overview.



Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming

State:	**Maryland**
Population:	4781468
Families:	1256327
Households:	1749342
Male %:	48.5%
Female %:	51.5%
Urban %:	81.3%
Average Age:	33.1
HS Diploma %:	78.4%
College Degree %:	31.7%
English Speaking %:	84.3%
Average Commute Time:	33
Carpool Commute %:	15.2%
Public Transportation %:	8.1%
Per Capita Income:	17730
Median Family Income:	45034
Median Household Income:	39386
No Income Households %:	15.3%
Average Persons per Family:	3.81
Average Workers per Family:	1.88
Housing Units:	1891917
Vacancy %:	8.2%
Average Bedrooms:	2.73
Average Persons per Unit:	2.73
Median Value:	115500
Median Mortgage:	919
Median Rent:	548
Rent % Household Income:	25.4
Flag Description:	The Maryland flag contains the family crest of the Calvert and Crossland families. Maryland was founded as an English colony in 1634 by Cecil Calvert, the second Lord Baltimore. The black and Gold designs belong to the Calvert family. The red and white design belongs to the Crossland family.

State:	**Massachusetts**
Population:	6016425
Families:	1525198
Households:	2244406
Male %:	48.0%
Female %:	52.0%
Urban %:	84.2%
Average Age:	34.5
HS Diploma %:	80.0%
College Degree %:	34.5%
English Speaking %:	79.0%
Average Commute Time:	28

Figure 4.4: Overview and detail

Bi-directionality is an important design property of coordination that is often violated in the design of user interfaces for other systems. Interface designers often neglect to include the latter direction of the overview-and-detail coordination. For example, many web pages with frames enable users to select an item in an overview frame to navigate the main frame (see web frames example in Chapter 1). However, when manually navigating the main frame, the highlight does not update in the overview. As a result, the interface can depict an inconsistent state, leading to user disorientation and confusion.

An advantage of Snap is that user interfaces constructed with it automatically inherit the robust nature of the Snap model, preventing such poor designs.

4.3.2 Propagation

When users invoke an action in a visualization, the effects will propagate across chained coordinations. All visualizations coordinated to that visualization either directly or indirectly through other visualizations will have their tightly-coupled actions executed. In the file-folders example, selecting a folder in the Hyperbolic Tree will highlight that folder in the scatter plot and load its files into the tabular visualization.

4.4 Additional Features

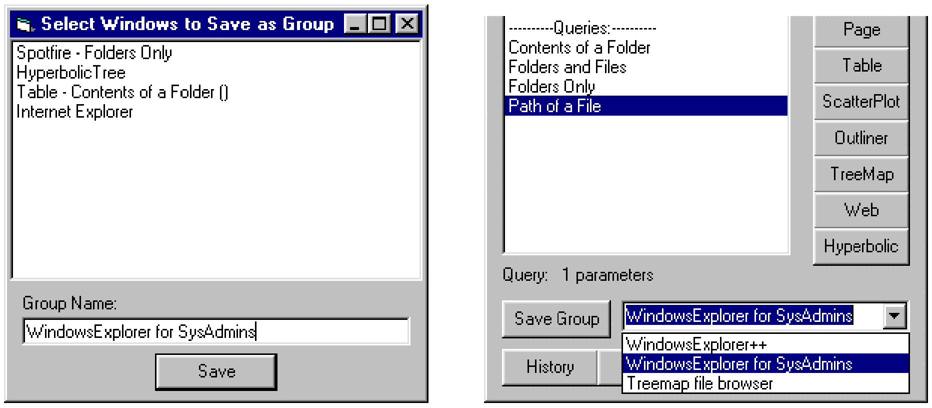
The Snap model and architecture also enable a variety of other user interface capabilities that magnify the utility of its coordinated-visualization interfaces. The following features have already been built:

4.4.1 Save Groups

A coordinated-visualization interface can be saved for later reuse, sharing, or distribution with data. The Save Group button on the Snap Menu displays the Save

Group dialog (Figure 4.5). Users select the visualizations they want to save by clicking the snap buttons on the visualizations, then assign the group a name, such as “Windows Explorer for System Administrators”.

Then, selecting that name from the combo-box on the Snap Menu automatically reconstructs the coordinated group of visualizations.



Group Name: WindowsExplorer for SysAdmins

Figure 4.5: Save Group dialog, and Snap Menu opening a group

4.4.2 Extract

Users explore information so that they can extract the needed information required to accomplish some other task, such as writing a report on Maryland’s economic status. Snap allows users to drag-and-drop tuples from visualizations into other applications such as Microsoft Word or an email message window. For example, users could select Montgomery County on a scatterplot of census data, and drag it to a Word document. When tuples are dropped, Snap displays a small popup list of the attribute names for the tuples (Figure 4.6). Users select the desired attributes, such as county name, per capita income, and population. Then Snap inserts the tuples values for those attributes into the document text.

Hence, snap can enable drag-and-drop for visualizations that do not support that capability because Snap tracks the selection actions and provides a drag initiation point in the visualization's snap button. Clearly, this capability would be significantly more powerful if multiple selection were enabled. Users could select the 10 most populated counties and extract their data to a document with a single drag-and-drop.

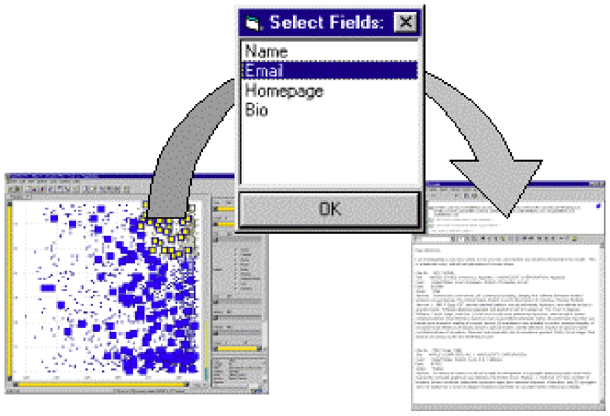
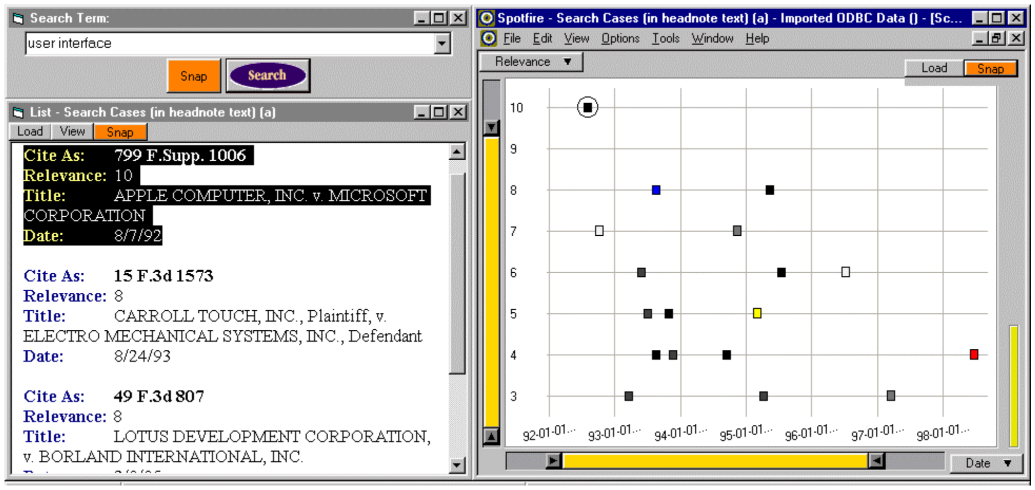


Figure 4.6: Attribute selector for drag-and-drop data extraction

4.4.3 Search Box

Snap provides a search box that can be coordinated to other visualizations. The search box enables users to directly type in a primary-key value, and initiate a coordination using that value. For example, if the folders primary-key values were their pathnames, users could coordinate the search box's *search* action to the *select* action of the folders scatter plot. Then typing a folders pathname into the search box will highlight the folder in the plot.

In a more advanced scenario, the search box can be used with computed joins. For example, a query could return all the files that have a provided key word in their name (or perhaps in their contents). The key word can be thought of as a foreign key, joined to a relation of keywords. The query can be opened in a list visualization and coordinated to the search box. The search box provides the key word, and the list displays the resulting file 'hits'. This approach is used in the WestLaw scenario for searching case-law documents (Figure 4.7). Typing a search term reveals case-law documents containing that term in a textual list of hits as well as a scatter plot. Notice that the query also returns a relevance value for use on the Y-axis of the plot. This interface could be coordinated to a case-viewer interface for examining hits.



Cite As: 799 F.Supp. 1006
Relevance: 10
Title: APPLE COMPUTER, INC. v. MICROSOFT CORPORATION
Date: 8/7/92

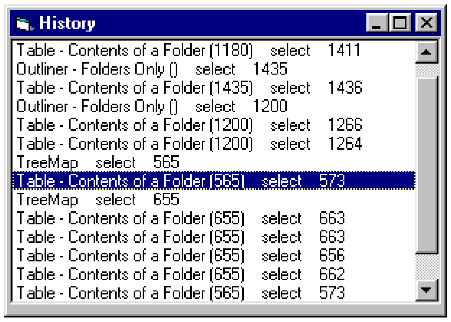
Cite As: **15 F.3d 1573**
Relevance: 8
Title: CARROLL TOUCH, INC., Plaintiff, v. ELECTRO MECHANICAL SYSTEMS, INC., Defendant
Date: 8/24/93

Cite As: **49 F.3d 807**
Relevance: 8
Title: LOTUS DEVELOPMENT CORPORATION, v. BORLAND INTERNATIONAL, INC.

Figure 4.7: Searching case-law documents

4.4.4 History

History keeping is becoming an important new research topic for user interfaces. History allows users to quickly review previous states when exploring. Since Snap receives action events from visualizations, Snap can easily keep a history list of all the actions users invoke. Snap's History window displays that list in chronological order (Figure 4.8). Each event indicates the visualization, action, and tuple. Selecting an event from the list re-invokes that action on that tuple in that visualization.



```
Table - Contents of a Folder (1180)   select   1411
Outliner - Folders Only ()   select   1435
Table - Contents of a Folder (1435)   select   1436
Outliner - Folders Only ()   select   1200
Table - Contents of a Folder (1200)   select   1266
Table - Contents of a Folder (1200)   select   1264
TreeMap   select   565
Table - Contents of a Folder (565)   select   573
TreeMap   select   655
Table - Contents of a Folder (655)   select   663
Table - Contents of a Folder (655)   select   663
Table - Contents of a Folder (655)   select   656
Table - Contents of a Folder (655)   select   662
Table - Contents of a Folder (565)   select   573
```

Figure 4.8:   Snap's History window

4.4.5 Shopping Basket

While exploring, users can easily gather a set of interesting tuples in Snap's Shopping Basket window, similar to the History window. Selecting a tuple in the basket also selects it in the visualization it originally came from. This allows users to collect a temporary set of items of interest while exploring. These can be used as bookmarks to return to those items in the visualizations.

4.5 Enhancements

The results of the user studies (Chapter 6) demonstrate that users, with some training, are able to construct coordinated-visualization interfaces with Snap. The studies also helped to identify improvements to the user interface that could dramatically reduce the need for training, improve user performance, and decrease error rates. These enhancements focus on reducing the need for query editing, and providing diagrammatic user interfaces that closely match the Snap model.

4.5.1 Automatic Query Generation

The study on construction revealed that creating new queries using Access was the primary difficulty for users. Reducing the need to create queries in this manner would be a major benefit. While the capability for creating queries enables generality, shortcuts are possible for the common simple situations. An applicable HCI design principle is: make common tasks easy, possibly at the expense of making rare tasks more difficult. There are two types of common simple queries that users must often create: selection, projection.

4.5.1.1 Selection

When using the *load* action in a coordination, Snap can automatically create the appropriate selection query based on the join relationship. In the file-folders example, when coordinated folders to files with *select* to *load*, Snap could automatically infer the SQL query for files based on the data schema:

```
SELECT * FROM files WHERE files.parentFolderID = <parameter>
```

4.5.1.2 Projection

Projection queries are often needed to extract certain attributes from a relation for display in a visualization. For example, an overview list of states' names is generated using a projection query to extract the Name field from the states relation:

```
SELECT id, name FROM states
```

In a modified Snap Menu, the tables and queries list is changed to an outliner control (Figure 4.9). Users can expand a table or query to reveal its attributes. Then, users can directly select attributes of a relation and drag them to visualizations, in addition to the capability to select an entire relation. Snap can automatically generate the projection query for the chosen attributes, and automatically include the primary-key attribute.

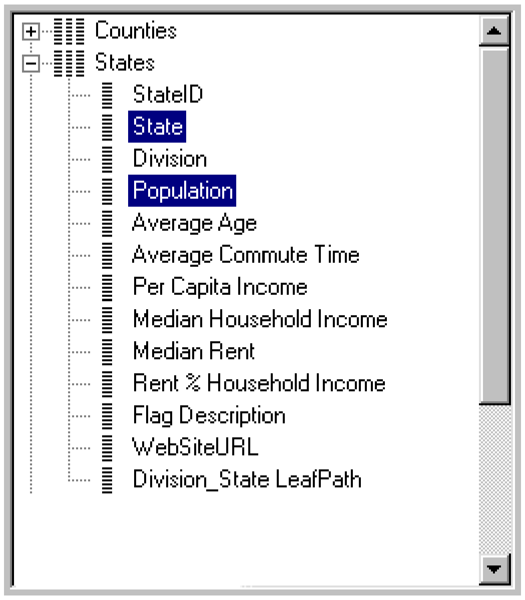


Figure 4.9:   Including attributes in the Snap Menu list of tables

4.5.2 Data Compass

The two-phase approach to coordination construction (opening visualizations, then specifying coordinations) can be combined into one. Once the user has initially opened a relation in a visualization, the Data Compass user interface displays which relations the user could coordinate to the current visualization based on the data schema. Users can select one of the relations, a visualization to display it in, and the actions to tightly couple. The new visualization is immediately displayed and coordinated to the current one as specified. For example, after displaying a visualization of the folders relation, the Data Compass indicates that users could coordinate a visualization of files, another visualization of folders, or other relations such as HardDrives or Users who own the folders (Figure 4.10).

This approach helps guide users in the construction process, and hence may significantly reduce training time. It may also match users' mental model more closely: "Where can I navigate to from here?". This may be valuable when exploring databases with many relations and very complex schemas (as in SeeData [AEP96]).

The Data Compass user interface divides relations that can be chosen into three groups based on the join relationship with the current relation:

- **Parents**: One to many towards the current relation.
- **Siblings**: One to one.
- **Children**: One to many towards the other relation.

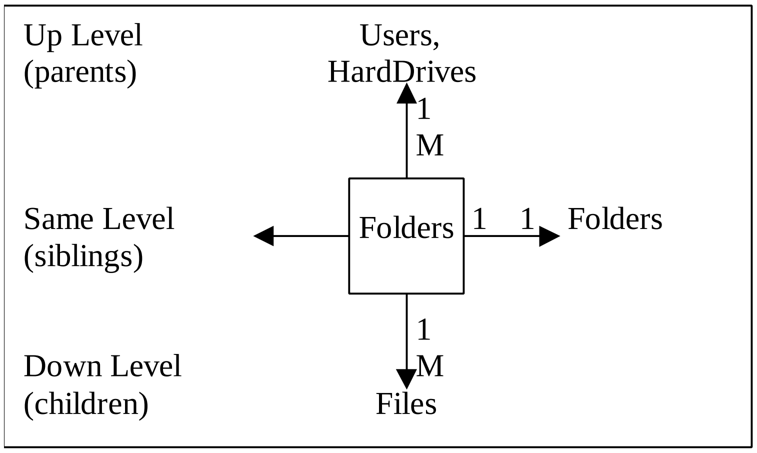


Figure 4.10: Data Compass

4.5.3 Overview Diagram

When users coordinate three or more visualizations, an overview diagram (Figure 4.11) is needed to help users understand and manage the coordination graph (that is, a visualization of visualization-coordination, or a meta-visualization). This helps to make the underlying Snap model more salient to users. The overview displays visualizations as nodes and coordinations as edges. Using direct manipulation, users can construct, edit, and delete coordinations. A debug mode can allow users to slowly step through a coordination propagation cycle. This diagram might also integrate the data schema to show the correspondence between relational concepts and Snap user-interface concepts, and to localize all interaction related to construction to a single window.

In LinkWinds [JBO94], users can temporarily view the linkages between its windows. When clicking the LinkWinds icon, it draws lines between the windows on the desktop.

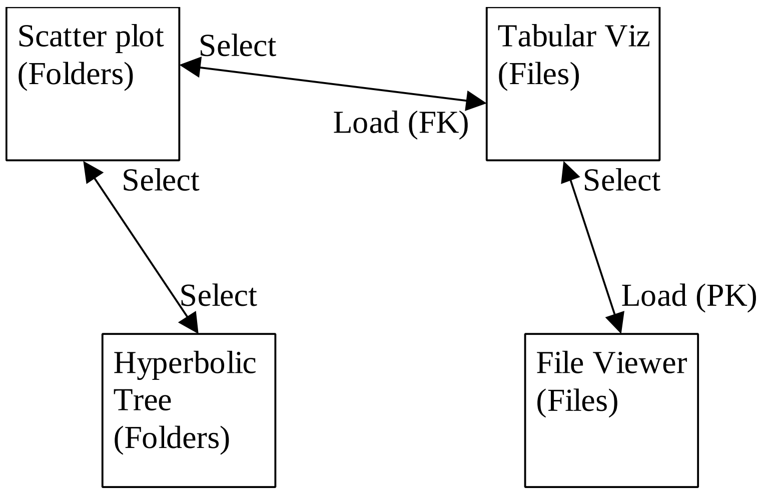


Figure 4.11: Overview diagram

4.5.4 Window Management

The studies also indicated that window management is a major burden for users. Two forms of automatic window management [KS97] can help:

- **Tiling**: When displaying many visualizations, users need to tile many windows on the screen. With tiling, users can easily "dock" visualizations to each other, so that resizing and rearranging is quick.
- **Packaging**: Users can package several visualizations into a single window using frames. This allows the group to be manipulated as a whole, for opening, coordinating, moving, resizing, and deleting. For example, in the WestLaw scenario, the case viewer is a saved group composed of three visualizations. This group is a semantic unit that can be instantiated and coordinated to other visualizations to load and display cases. The three visualizations could be packaged to reflect this grouping. This allows saved groups to be treated as a single visualization, turning composites into new primitives.

4.6 Summary

The Snap user interface enables users to explore information by quickly constructing coordinated-visualization interfaces without programming. Users first open relations into visualizations, then coordinate them by selecting actions to tightly couple. Snap also enables a host of additional features that further amplify its value. Several enhancements to the Snap user interface have also been described based on results from user studies in Chapter 6.

Chapter 5:
Software Architecture for Visualization Coordination

The Snap-Together Visualization software architecture enables the dynamic construction of coordinated visualizations, providing flexibility in data, visualizations, and coordinations. A major goal of Snap is to coordinate independent visualization tools. The Snap architecture implements the Snap model and exploits existing functionality of visualizations to accomplish this goal. Because of Snap's clean design, researchers and developers can easily snap-enable their independent visualization tools, allowing users to employ the tools in coordinated-visualization interfaces of their own.

5.1 Architecture Overview

The Snap system acts as a centralized intermediary between visualizations (Figure 5.1). It also mediates between the database and the visualizations. The Snap architecture insulates visualizations from each other, the database, and the rest of the system. This protects visualizations from having to be programmed to handle the complexities of visualization coordination. In fact, visualizations are completely unaware of the concept of coordination. Their only connection to Snap is through a very simple API (application programming interface).

This is different from standard approaches in fully integrated systems. For example, in the Visage [RLS96] architecture, when users highlight an item in one visualization it broadcasts a message to all other visualizations. Then each visualization must itself determine the relationship of that item to its set of items and calculate what action to take.

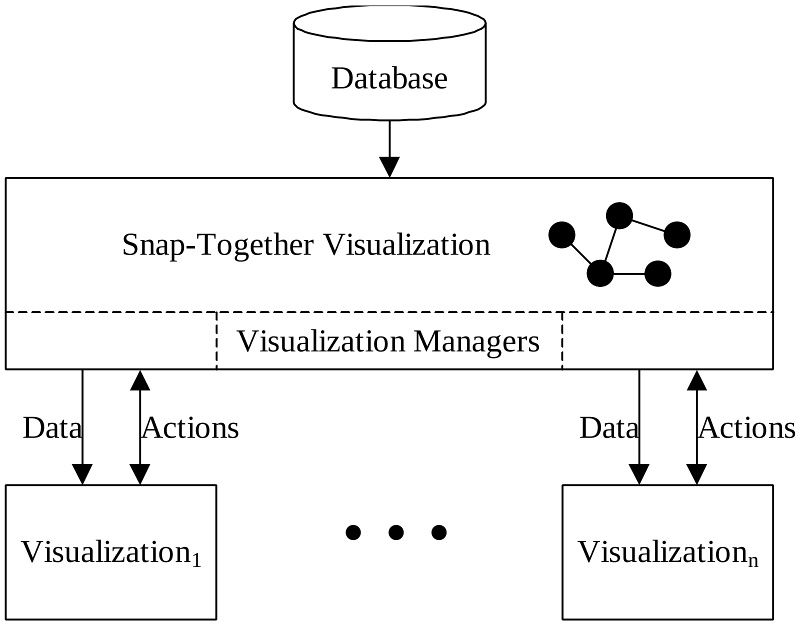


Figure 5.1: Snap's software architecture

5.2 Visualizations

At start up, Snap's Main Menu displays a menu of available visualization tools. Each visualization must initially register with Snap in order to be included in this menu.

When initially opening a database, Snap extracts schema information from the database, including the list of relations (tables and queries) to display in the Main Menu.

When users open a relation into a visualization, the following operations execute:

1. Within Snap, a Visualization-Manager object is instantiated to handle communication with the visualization.
2. The visualization tool is instantiated.

3. If the relation is a query, the query is executed in the database.
4. The data in the relation is loaded into the visualization, using the visualization's Load Procedure (described in the API section below).

5.2.1 Goals for Snap-Enabling Visualizations

Snap is designed to be open, so that developers can easily make their independent visualization tools snap-able, including existing visualizations and newly developed visualizations. The effort required to snap-enable an off-the-shelf visualization is minimized to the extent that even a developer who is not the original implementer of the tool should be able to make the necessary modifications.

To accomplish this, snap minimizes the impact on visualization implementation. Snap uses a simple API (application programming interface) to communicate with visualizations. This is analogous to API's in modern window-management systems for utilities such as cut-and-paste or drag-and-drop. The Snap API is proposed as a similar standard, that can be easily added to a visualization tool by its developers, enabling users to immediately snap it with many other visualizations. This greatly increases the value and usefulness of the tool for little cost. Effort is low and payoff is high.

Snap limits programming effort by exploiting existing functionality of visualizations to coordinate them together. The functionality of typical visualization tools includes the ability to load a data set (e.g. from a file) and display it as visual items in a window. These tools often provide some form of interactivity, allowing users to select items or navigate between items.

To maximize the compatibility of the architecture with typical visualization tools, and minimize the effort to integrate these tools, the architecture places upper limits on visualization requirements. Primarily, these aspects of visualizations are NOT changed:

- Remain independent software entities. Run as stand-alone applications as normal, and are not compiled into Snap.
- Themselves determine what actions they support (e.g. select, scroll).
- Use their existing data input format.
- Do not need to deal with the larger data context of the database. Handle the data loaded into them by Snap as normal.
- No new user interface requirements.
- No requirements for shared data structures, etc.
- Do not need to be made aware of the database, other visualizations, or coordination.

Furthermore, Additions to the visualizations are limited to:

- Simple communication protocol.
- Identify tuples by primary-key only (e.g. no complex attribute processing).

5.2.2 Snap Button

When instantiating a visualization, the Visualization Manager automatically adds the snap button Snap to the visualization's user interface. This is similar to the concept of window managers adding window decoration and controls to each window when opened. This provides an interaction point for the user for each visualization, and is used for coordination construction, loading different data, saving groups, etc.

Ideally the snap button would appear next to the minimize, maximize, and close buttons on the window's title bar. However, due to minor implementation constraints, the snap button is placed just below these buttons within the window's client area. This is accomplished by simply inserting a small child window containing only the snap button into the visualization window. Hence, the snap button moves and overlaps with the visualization's window. The Visualization Manager tracks resize events of the visualization's window, and adjusts the position of the snap button within the visualization accordingly.

This approach saves developers from needing to add Snap user interface functionality to their visualizations.

5.2.3 Visualization API

To be snap-enabled, each visualization must implement the following API. Snap communicates with the visualization by connecting to these entry points on the visualization. It is worth noting that this API is not necessarily specific to visualization coordination. It is quite general, and could be useful for many other applications such as history keeping, end-user programming, multi-user collaboration, etc.

There are only three elements in the API:

5.2.3.1 Load Procedure

Procedure **doLoad**(filename **|** dataObjRef **|** SQLstring, PKattribute)

Snap can invoke this routine to load data into the visualization. Visualizations can choose one of three methods to receive the data:

- **File:** Snap writes the tuples to a temporary file in the format expected by the visualization, using a translator routine. This is the approach for most typical research visualization tools, such as Treemaps and Hyperbolic Trees.
- **Memory:** Snap provides the data using standard ODBC data objects (Microsoft DAO or ADO). This is common for visualizations that were developed specifically as components (e.g. ActiveX), such as the tabular visualization which uses a standard grid control, or developed specifically for Snap, such the textual list visualization.
- **SQL:** Snap provides the ODBC connect string and the SQL query string that the visualization then uses to extract the data from the database itself. This is useful for visualization tools that have built in database support, such as Spotfire.

Visualizations may also need to know which attribute to use as the primary key. Visualizations should attempt to preserve any visual settings across loads.

5.2.3.2 Action Procedure

Procedure **doAction**(action, PKvalue)

Snap can invoke this procedure to programmatically execute the specified action on the tuple identified by the specified primary-key value. For example, a coordination could invoke the *select* action on a Spotfire scatter plot to highlight the specified dot. Each visualization publishes the list of actions it supports to Snap at registration time.

5.2.3.3 Action Event

Event **onAction**(action, PKvalue)

The visualization triggers this event to Snap whenever users perform one of the visualization's supported actions on a tuple. The visualization reports the action name and the primary-key value of the tuple. For example, when users click on a dot in a Spotfire scatter plot, Spotfire reports the *select* action.

5.2.4 Visualization Registration

At registration time, each visualization specifies its:

- Name: for identifying it to the user, as on the Snap Main Menu.
- Description: more detailed text.
- Launch string: specifies how Snap instantiates the visualization.
- List of actions. each action is a string, for identification to the user in the Snap Specification dialog and for use in the API.
- Load method: File, Memory, or SQL (see API Load Procedure).

Ideally, developers could register their visualizations with Snap using a registration user interface to a registration database.

5.2.5 Programming Effort

Adding Snap's API to a visualization requires only a small amount of code. First, there may be some initial overhead in enabling the visualization for communication. In the current implementation, this means making the visualization into a COM object. Fortunately, the popular development tools can do this automatically.

Second, the three API elements must be implemented. Since a visualization already has functionality to load data, the Load Procedure can simply call that existing code. Hence, this is quite simple to add, requiring essentially two lines of code (the procedure declaration and the call). Likewise, the Action Procedure can use the existing user interface code to perform actions. However, additional code may be needed to search internal data structures to locate the item identified by the given primary-key value. Corresponding code that searches for items based on user events (mouse clicks) can be copied and modified. This usually requires 2-10 lines of code. The Action Event simply requires adding the event trigger in the appropriate callback routine of the visualization's user-interface code, requiring one line of code per supported action. Also, the data structures may need to be expanded slightly to support the storage of the primary-key values.

Finally, a translator procedure may be needed that converts the input data from the memory format (relational data objects) to the input format of the visualization tool. However, this could be claimed 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 Snap they need at most one: to convert their data into a relational database. And visualization developers need to supply only one: to convert the relational format to their visualization's format.

To snap-enable the Treemap visualization tool, which was originally developed by others, required approximately 2 hours of work to add approximately 20 lines of code to its software (using Borland Delphi's Object Pascal).

In some cases, access to the source code is not necessary. Some well-designed component-based visualizations, such as Spotfire (commercial software), already support a full suite of methods and events. A simple wrapper program can be written in Visual Basic (VB) that translates the Snap API protocol to calls to the visualization component. Snap provides a template wrapper. Snap-enabling Spotfire required approximately 10 lines of VB code.

Java and web-based applications can be enabled using Internet Explorer (IE). For example, the Hyperbolic-Tree Java applet was enabled using a small VB wrapper to control IE, and a simple HTML page to control the Hyperbolic Tree applet using Javascript.

SAS JMP is an example of a visualization package that could not be enabled well. Its programming API has many methods, but no events (callbacks). Hence, the Load and Action Procedures could be implemented in the VB wrapper, but the Action Event could not. A request has been given to its developers to include action events.

5.3 Coordination

When users coordinate visualizations, snap maintains a graph data structure representing the visualizations and coordinations. Then, when users invoke an action in a visualization during coordination operation, the following execution takes place (see Figure 5.2):

1. The visualization notifies Snap of the action and the primary-key value of the tuple acted on, via its Action Event.
2. Snap begins a traversal of the coordination graph starting at that visualization and action.
3. For each visualization encountered in the traversal, Snap invokes the tightly coupled action on the visualization:

a. If the action is not a load action, then the action is programmatically invoked directly on the visualization, passing the primary-key value as parameter, via its Action Procedure.

b. If the action is load, then the Visualization Manager executes the selection query using the primary-key value as the query parameter, and loads the results into the visualization via its Load Procedure.

For example, in the file-folders example, when users select the folder with primary-key value "MyDocs" in the scatter plot, then Snap calls on the Hyperbolic Tree to select "MyDocs". Then for the tabular visualization, Snap executes and loads the results of the query:

```
SELECT * FROM files WHERE files.parentFolderID = "MyDocs"
```

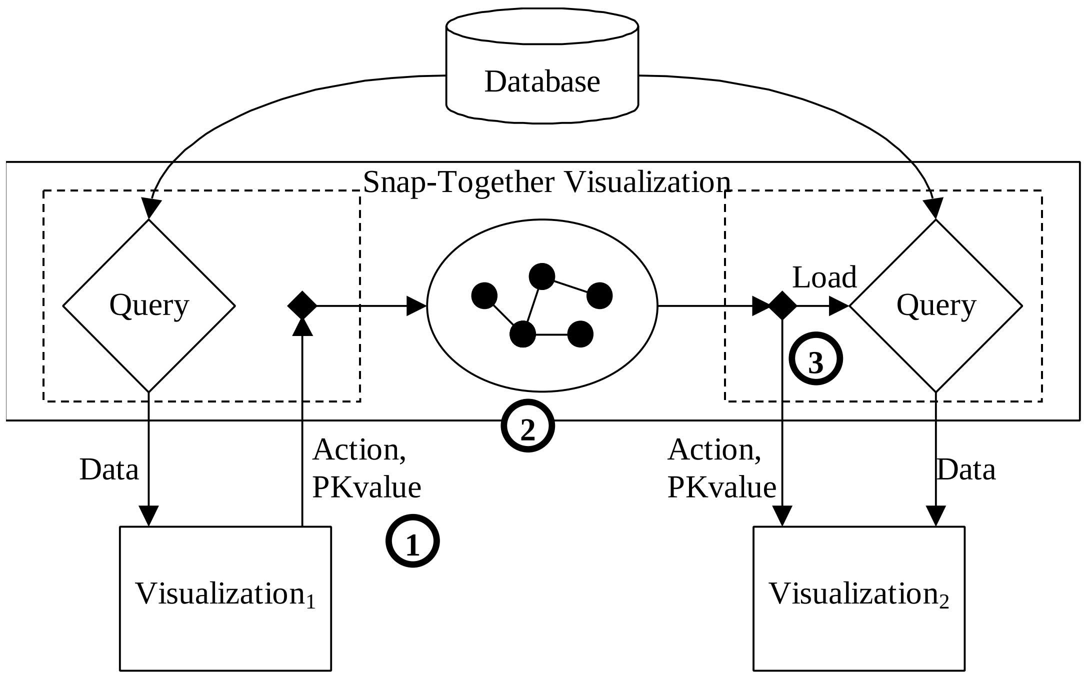


Figure 5.2: Coordination Operation

The coordination graph data structure and coordination propagation traversal algorithm provide the generality that makes the Snap architecture flexible for visualizations and coordinations (e.g. third level of flexibility). Users can construct any possible combination of visualizations and coordinations as needed.

5.3.1 Data Structures

The coordination graph data structure is based on the Snap model, and is composed of a list of the currently open visualizations and a list of the currently constructed coordinations. As users construct or delete visualizations and coordinations, Snap adds and removes from these lists.

```
Coordination Graph data structure:
        List of Visualization structures
        List of Coordination structures

Visualization data structure:
        Visualization object reference
        Relation name
        List of boolean marks for each action (used during propagation)

Coordination data structure:
        Pointer to Visualization1 structure
        Action1
        Pointer to Visualization2 structure
        Action2
```

5.3.2 Algorithm

The coordination propagation traversal algorithm executes the tight couplings and implements the transitivity property of the Snap model. During coordination operation, when users invoke an action on a visualization, a depth first traversal of the coordination graph is initiated:

```
Procedure beginPropagation(visualization, action, PKvalue)
    Clear all action marks in all visualizations
    Call traverse(visualization, action, PKvalue)


Procedure execute(visualization, action, PKvalue)
    If not marked (visualization, action) then
        If action = load then
            Execute visualization query(PKvalue)
            Call visualization.doLoad(query results)
        else
            Call visualization.doAction(action, PKvalue)
        Call traverse(visualization, action, PKvalue)


Procedure traverse(visualization, action, PKvalue)
    Mark visualization, action
    For each coordination in graph.coordinationList do
        If visualization = visualization1 and action = action1 then
            Call execute(visualization2, action2, PKvalue)
        Else if visualization = visualization2 and action = action2 then
            Call execute(visualization1, action1, PKvalue)
```

5.4 Issues and Tradeoffs

5.4.1 Independent vs. Integrated Visualizations

The Snap architecture is designed to use independent visualization tools. An alternate approach would be to fully integrate visualizations by custom implementing them within the context of the coordination system (as in Visage [RLS96], DEVise [LRB97], Spotfire, etc.). Each approach has corresponding advantages (+) and disadvantages (-):

Independent Visualizations	**Integrated Visualizations**
+ Open system, others can easily add visualizations	- Closed system, only system developer can add visualizations
+ Reuses existing visualizations from the field	- Popular visualizations must be re-implemented within the system
+ Visualization development unaffected	- Visualizations must use designated structures
+ Visualizations can be used outside the system	- Visualizations only work within the system
+ Clean component-based design, visualizations insulated via API	- Potential inter-dependency complexities
+ Consistent coordination model	- Potential coordination inconsistencies
- Use only existing functionality of visualizations	+ Can add new functionality to visualizations
- Visualization user interface inconsistencies	+ All visualizations implemented with same look and feel
- Potential performance hit	+ Potential performance boost from shared data structures, etc.
- Static coordination model	+ Can add advanced custom functionality for coordinating dynamic data, edits, etc.

The Snap architecture employs a component-based approach, in which visualizations are implemented as individual units rather than integral to monolithic systems. This programming approach is becoming increasingly popular in commercial visualization and other domains due to benefits of modularity, reuse, etc. For example, AlphaBlox [IDC99] enables rapid deployment of web-based analytical applications by dropping visualization and data components into web pages.

While Snap works well to coordinate full-fledged feature-rich visualization applications such Spotfire, the Snap approach steers developers towards implementing smaller simple visualization components as in the Hyperbolic Trees applet. This eliminates some of the extra visual clutter of toolbars and menus for each visualization.

5.4.2 Effort vs. Payoff

In the design of the Snap API, part of the goal is to maximize benefits while minimizing effort required by visualization developers. A larger and more complex API would enable more functionality (e.g. coordinating dynamic data, edits, etc.), but would require more effort for visualization developers and the disadvantages of the integrated approach begin to creep in. Hence, when increasing effort, the law of diminishing returns results in reduced payoff. I believe that Snap finds the sweet spot where effort is low and payoff return is maximized.

5.4.3 Snap vs. Programming

When constructing coordinated-visualization interfaces, one can either use Snap (visualization or coordination flexible) or program the interface by hard coding the desired coordinations between visualizations (data flexible or non-flexible). Each approach has corresponding advantages (+) and disadvantages (-):

Snap	**Programming**
+ Non-programmers (for enabled visualizations)	- Programmers only
+ Quick and easy	- Time consuming and difficult
+ Can make throw-away solutions for temporary or short-term needs	- Short-term needs go unmet
+ Interfaces are changeable on the fly	- Static, inflexible, slow turn-around
+ Can prototype many options	- Prototypes typically non-functional
+ Robust coordination model	- Prone to mistakes, inconsistencies
+ Guided by Snap model	- Design from scratch
+ Once enabled, visualizations are reusable in many different interfaces	- Visualizations hard-coded each time
- Potentially disparate visualizations	+ Package in custom user interface
- Bounded functionality	+ Custom functionality as needed

How much effort is saved by using Snap instead of programming a hard-coded coordinated-visualization interface by hand? It is difficult to measure the number of lines of code saved because it is not clear what code in the hard-coded interface to count. What would the programmers be starting with? Snap provides a total solution from data to coordinated-visualization interface, that covers a lot of functionality.

Yet, even more than the number of lines of code is the significant amount of consideration and care programmers must employ. Implementing a coordinated-visualization interface is very tricky. An interface with two coordinated visualizations may be straightforward, but complexity quickly increases with the number of visualizations and coordinations.

An examination of the Snap's functionality reveals the amount of complexity that programmers must consider when implementing a coordinated-visualization interface. First, programmers must consider the design of the coordination. The Snap model provides significant guidance to how the coordinations work. Programmers must implement affordances for actions. Visualizations must be able to notify of user actions and invoke and respond to actions programmatically. A method is needed to uniquely identify data items. Actions must be propagated to other visualizations. Functions are needed to relate data items between visualizations. Programmers must keep track of which visualization initiated the action, ensure that each action in each visualization propagates to all others as needed, ensure that programmatically invoking actions doesn't generate new actions, and ensure that each action gets invoked on a visualization only once. Finally, data handling is needed for processing, subsetting, and loading data into visualizations, possibly as a result of coordinations.

Naturally, this process is prone to errors, bugs, incomplete implementations, and inconsistencies in design. For example, many web designers fail to include bi-directionality in overview-and-detail coordinations between frames. Selecting an item in the overview highlights the item and displays corresponding details in the detail frame. However, navigating the detail frame using the scroll bar or next/previous buttons on the page fails to update the highlight in the overview. This results in inconsistent state and confusion.

Whereas, Snap opens design capability to non-programmers. This enables the construction of coordinated-visualization interfaces in many situations where a requirement for programming would immediately prevent its use. Snap does not require programming savvy, development tools, knowledge of the visualizations' implementation, etc.

5.4.4 Scalability

The Snap architectural approach of using independent visualizations has a potential disadvantage in system performance. In direct-manipulation environments, user actions should result in visual feedback within 100 milliseconds [Shn98]. Hence, in coordinated-visualization interfaces, propagated actions should occur within 100 milliseconds from the user action invocation. In an integrated approach, all visualizations can be implemented to use shared data structures and optimized for coordination operation. However, with independent visualizations, each visualization instantiates its own potentially-large data structures and may not have been implemented from the perspective of coordination. This could mean that programmatically invoking actions on visualizations is slow.

In general, displaying several visualizations simultaneously is not a problem for memory and swapping. Modern systems are designed to handle many open applications and windows. Screen space is the limiting factor here. However, if invoking actions on visualizations is slow, then the number of open visualizations may serve to multiply that delay. Furthermore, a coordination propagation is only as fast as the slowest visualization involved in the propagation. By default, in the COM implementation, API calls are blocking. This means that while an action invocation is executing, Snap is stalled. There are two potential bottlenecks in the API: the Action Procedure and the Load Procedure.

The Action Procedure may have to perform a search on the visualization's internal data structures to locate a tuple by its primary-key value. For naively implemented visualizations, this requires an O(n) search. For example, Spotfire's VB wrapper executes an O(n) search using Spotfire's programmer API. Performance tests on a 300 Mhz Pentium computer measures this search at about 1 second per 1000 tuples. This can be vastly improved using hash tables or other data structures to map primary-key values to Spotfire data-structure indices or pointers.

A potential solution to this problem would be for Snap to manage hash tables for each visualization. After loading a relation, a visualization could perform a single traversal of its internal data structure, reporting each primary-key value and internal pointer pair to Snap. Snap could store these in a hash table. Then, when invoking an action on a visualization, Snap could provide a direct pointer to the tuple.

The Load Procedure is used to initially load data into a visualization. Slow performance here is acceptable. However, it is also often used to repeatedly load different data into a visualization during a drill-down coordination. Fast performance in this case is needed to enable users to quickly explore aggregates. Again, testing Spotfire (a known slow loading program) on the same computer with the web log data (about 25,000 tuples, 10 attributes), Spotfire loads approximately 1000 tuples per second with a minimum of about 1 second. Hence, displaying the whole relation is a significant delay. However, in a drill-down coordination, only a fraction of the data is loaded. Users can explore a million tuple relation using aggregation and drill-down, by displaying 1000 aggregates in one Spotfire plot and 1000 tuples of a selected aggregate in another plot. That results in a 1 second delay for each aggregate.

When dealing with large relations or slow visualizations, there are some potential solutions to help users avoid long unwanted delays:

- **Warning**: The textual list visualization displays a warning message if it attempts to load a relation of more than 200 tuples. Users have the option to cancel the load entirely.
- **Loose coupling**: Instead of loading immediately, a slow visualization could simply indicate that it has become out of date with respect to coordination. Then, users could manually trigger an update when desired.

Each of these could be implemented within Snap as a general solution. Users could control these options through the Snap user interface.

5.5 Implementation Details

Snap is currently implemented in the Windows platform. It is based on the Microsoft COM/ActiveX model for communication in the API. Visualizations are COM objects, exposing the visualization API as methods and events. Snap creates and controls visualizations using OLE automation. It uses the Windows API and the visualization's window handle to insert the snap button into each visualization and to track window resizing for saving and opening visualization groups.

Snap accesses ODBC databases using the Microsoft DAO object model. This allows Snap to extract schema information, execute queries, and extract data. Snap can reliably extract table and query information, but can retrieve join relationship information for only some database formats. Snap has been used with Microsoft Access and Oracle databases. For Access databases, Snap instantiates the Access GUI to allow users to edit and manipulate the database. For Oracle and others, Snap provides a simple SQL query text editor.

Snap is implemented in Visual Basic, an ideal environment for working with COM. There are four primary code modules (Figure 5.3):

- **Snap Menu**: implements the Snap Menu, and visualization registration.
- **Database Manager**: handles database access, querying, schema extraction.
- **Coordination Manager**: implements the Snap Specification dialog, coordination data structures and propagation algorithm.
- **Visualization Manager**: handles communication with visualizations, and implements the snap button. Instantiated for each visualization.

Additional modules handle the user interfaces and functionality for saving groups, history keeping, shopping basket, drag-and-drop data extraction, and search box. There is also the implementation of a few of the visualizations (text list, table, outliner) and wrappers for others. The compactly designed Snap code is on the order of 2000 lines of

VB code, not including user interface properties and layout definitions. The implemented visualizations and wrappers are an additional 2000 lines.

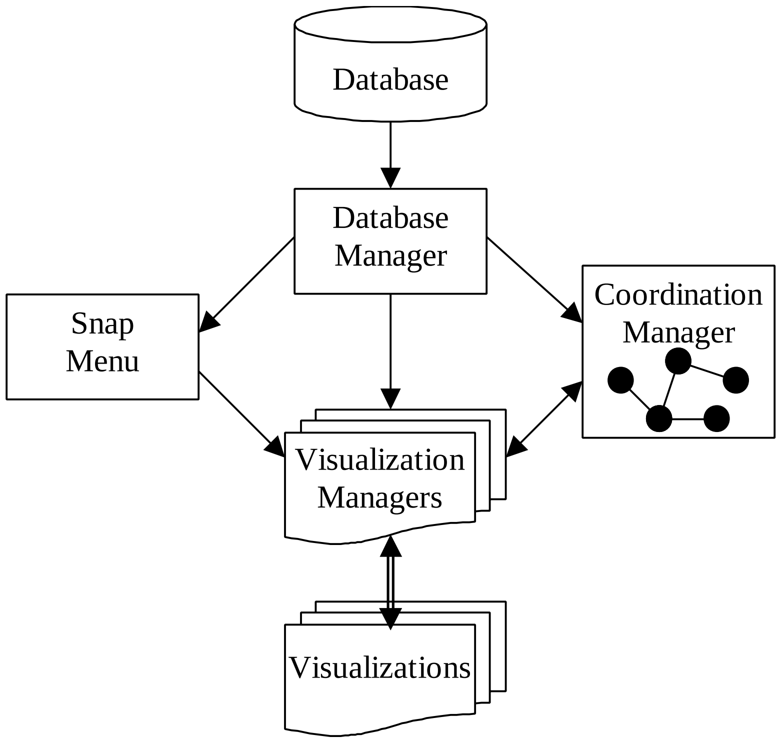


Figure 5.3: Software modules

5.6 Extensions

The Snap software architecture lays out a foundation on which several interesting extensions could be built.

5.6.1 Packaging and Deploying

One of the primary uses of Snap is to allow designers or data disseminators to construct coordinated-visualization interfaces for deployment to other users. Snap has the capability to save coordinated-visualization groups. But to truly enable deployment, a mechanism is needed to package saved groups as standalone executables.

Essentially, Snap could become the 'Visual Basic' of information visualization. Designers could quickly construct an interface making use of third party visualization components, and essentially compile it into an executable containing only the necessary visualizations and functionality for coordination operation.

Licensing issues with commercial visualizations could be handled in the same way that VB handles commercial controls. Designers purchase the visualizations, and can distribute them in their constructed interfaces. But users of the constructed interfaces, cannot switch to 'construction mode' (VB 'design mode') to make new interfaces with the commercial visualizations.

5.6.2 Collaboration

The Snap architecture provides capabilities that could support collaborative visualization. There are two forms of collaboration with respect to time: synchronous and asynchronous.

5.6.2.1 Synchronous Collaboration

Synchronous collaboration refers to multiple users working together at the same time. Often, the users are at different computers and locations.

When coordinating independent visualization tools with Snap, there is absolutely no reason why the visualizations have to be running on the same computer. Snap could be used to synchronize information exploration on multiple users' screens (similar to Suite [DC95]). One user could explore and point out interesting phenomena in the data while other users at remote locations watched. In fact, different users could use different visualizations according to their preferences (similar to RENDEZVOUS [Hil92]).

Snap provides a very efficient communication protocol that could easily be transported over the internet. In fact, COM already has support for remote procedure calls and distributed computing called DCOM. The Snap API could simply be invoked on visualizations running on remote machines.

5.6.2.2 Asynchronous Collaboration

Asynchronous collaboration refers to multiple users working together but at different times. Snap's capability for saving coordinated-visualization interfaces and history keeping could be used to support this type of collaboration too. The history keeping could be used to easily save the current state of exploration during coordination operation. This could then be published so that other users could see what has been discovered, similar to LiveDocs [MHG00]. In addition, the full history could be used to create animations of exploration for other users, as in SimPLE [PRR99]. For example, a professor could navigate through a scientific database to show several important phenomena, and then send out the history to students to replay for homework. Again, Snap provides a very efficient mechanism to save and distribute such histories along with the specification for the saved interface.

5.6.3 Dynamic Data Consistency

Some visualizations may allow users to edit the data, such as adding, deleting, or renaming a file in the file-folders example. Snap could be extended to coordinate data consistency between visualizations in the face of changing data. An additional procedure could be added to the API to notify of changes to individual tuples. Ideally, visualizations could reload only changed tuples without reloading the entire relation.

This capability might also enable the display of dynamic databases, as in stock market applications or air traffic control. If the data update rate is low (e.g. changing a few tuples per second), Snap could update visualizations with changing data values. However, further research is needed to explore specialized architectures that can scale up to high data update rates.

5.6.4 Integrating into Operating System

While the Snap architecture is currently implemented as a standalone application, it could be integrated into data systems or operating systems. For example, Snap could be integrated into the ODBC architecture in the Windows operating system. The Snap Visualization API could be adopted into the current ODBC API standard. Snap's GUI could become part of Windows, and the snap buttons part of the window decorations. Then, ODBC compliant applications could be used as snap-able visualizations.

This approach has several major benefits. ODBC benefits by adding this powerful new feature. Snap benefits by joining an existing strong standard and by potential improved performance due to integration. Visualizations benefit by simplifying development due to a single unified standard. This approach might also enable more applications, such as drawing from multiple distributed databases.

5.7 Summary

The Snap software architecture enables flexibility in data, visualizations, and coordinations. Its visualization API enables developers to easily snap-enable their independent visualizations. The data structure and algorithms are based on the sound Snap model. The architecture clearly demonstrates major advantages (and some disadvantages) over programming and the fully integrated approach. It provides a solid foundation for potent new future directions.

Chapter 6:
Evaluation of Coordination Construction and Operation

Studying the use of Snap is important for two reasons:

- To evaluate the usability and benefit of the Snap system itself and discover potential user interface improvements.
- To gain a deeper level of understanding about users' ability to understand, construct, and operate coordinated-visualization strategies in general.

Two separate studies were undertaken to evaluate two distinct aspects of coordination [NS00b]:

1. **Construction**: First, can users successfully construct their own coordinated-visualization interfaces?
2. **Operation**: Second, can users then operate the constructed coordinated-visualization interfaces to explore information beneficially?

6.1 Evaluation of Coordination Construction

The goal of the first study is to determine if users can learn to construct coordinated-visualization interfaces and how difficult it is for users to construct them, in terms of success rate and time to completion, and to identify cognitive trouble-spots in the construction process. Hence, this study examines the flexibility that Snap provides. Can users grasp the concept of coordinating two independent visualizations together to form a unified browsing tool? What cognitive issues are involved, how much training is required, how do users' backgrounds affect performance, and can relatively novice users construct powerful exploration tools in a short time? This study also reveals potential Snap user interface improvements.

The Snap-Together Visualization system is used to examine these issues. Currently, Snap employs a 2-step approach to constructing coordinated-visualization interfaces. First, users drop relations into visualizations. Second, users snap the visualizations together to coordinate actions between them. For this study, Snap uses Microsoft Access GUI to enable users to create and edit queries.

6.1.1 Procedure

Six subjects participated, one at a time. Four of the subjects were employees of the U.S. Census Bureau, three of whom were data analysts or statisticians, and one a programmer. The other two subjects were computer science graduate students on campus.

First, background information was obtained from each subject concerning their occupation and experience with: census data, computers, databases, Microsoft Access, visualization tools, and programming.

Then, each subject was trained on Snap-Together Visualization. The training program consisted of:

1. A quick demonstration of Snap by the administrator to give the subject an overview and motivation.
2. Review of various background concepts including:
    - Relational database concepts including: tables, records, fields, primary keys, foreign keys.

- Database query concepts including: projection, selection, sort, join.
- Snap model concepts.

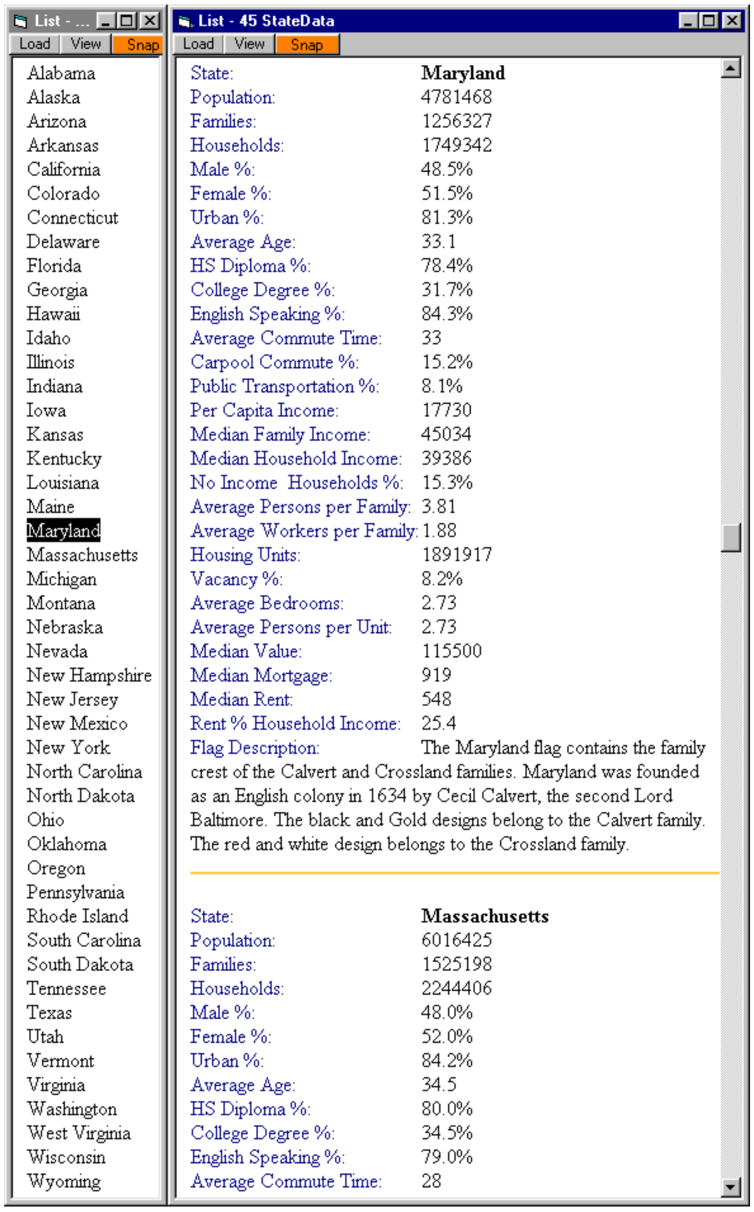
3. Detailed instruction on the use of Snap and Microsoft Access. The subjects walked through the construction of a few variations of coordinated-visualization interfaces for browsing census data. This demonstrated how to construct common types of coordinations.

Then, when confident to continue, each subject began the testing phase. Subjects were given a database of census data for the U.S. states and counties, and Snap (including a set of Visualization tools) and Microsoft Access. Testing consisted of three exercises in which subjects were asked to construct a coordinated-visualization user interface according to a provided specification:

Exercise 1: The first specification consisted of a printed screenshot of the desired user interface (Figure 6.1). The interface is a pair of textual visualizations with overview-and-detail coordination for browsing state data. This trial was designed to be fairly easy, to be similar to those constructed in the training, and to build confidence.

Exercise 2: The second specification was also a screenshot (Figure 6.2), but more difficult. It uses a textual list, Spotfire scatterplot, and tabular visualization to browse census data for states and counties. It involved a one-to-many join relationship, so that selecting a state would display data for that state's counties.

Exercise 3: The final specification consisted of a textual description of the browsing task that the constructed interface should support: "Please create a user interface that will support users in efficiently performing the following task: To be able to quickly discover which states have high population and high Per Capita Income, and examine their counties with the most employees." This trial was designed to test if subjects could think abstractly about coordination, think task-oriented, think in terms of user-interface design, and to allow for potential creativity and variation.

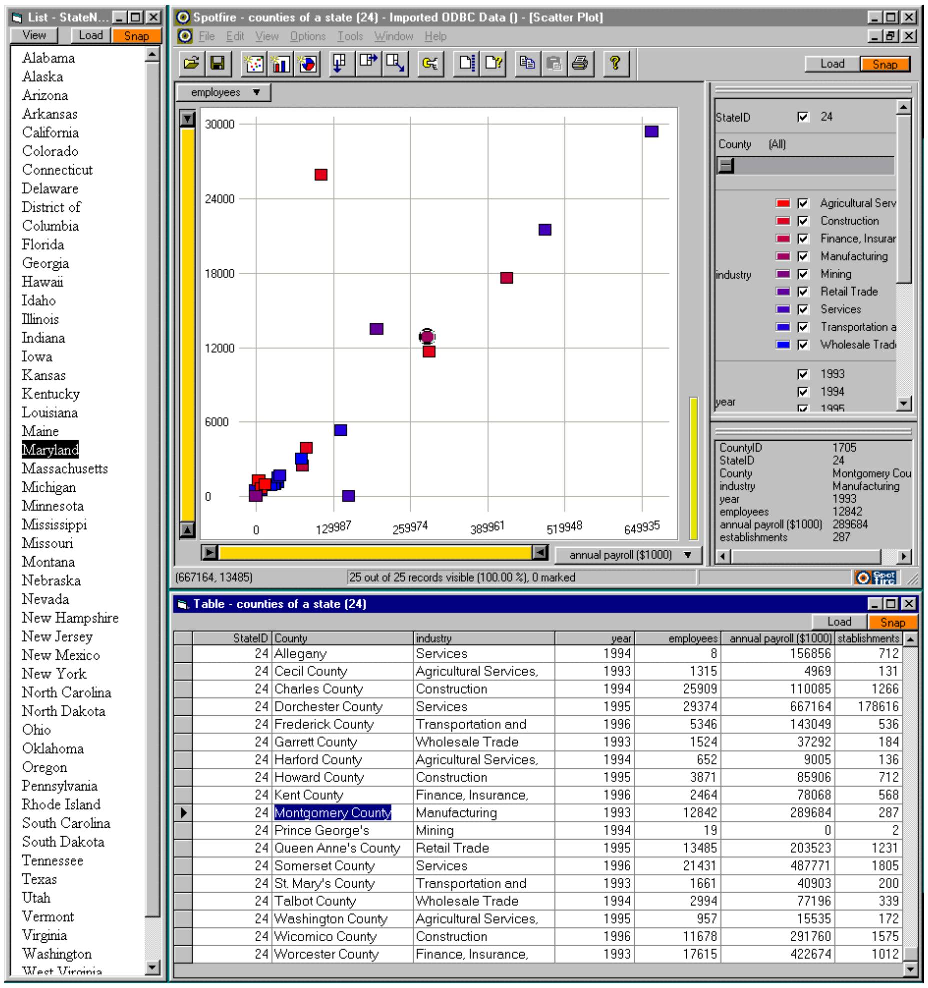


Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming

State:	**Maryland**
Population:	4781468
Families:	1256327
Households:	1749342
Male %:	48.5%
Female %:	51.5%
Urban %:	81.3%
Average Age:	33.1
HS Diploma %:	78.4%
College Degree %:	31.7%
English Speaking %:	84.3%
Average Commute Time:	33
Carpool Commute %:	15.2%
Public Transportation %:	8.1%
Per Capita Income:	17730
Median Family Income:	45034
Median Household Income:	39386
No Income Households %:	15.3%
Average Persons per Family:	3.81
Average Workers per Family:	1.88
Housing Units:	1891917
Vacancy %:	8.2%
Average Bedrooms:	2.73
Average Persons per Unit:	2.73
Median Value:	115500
Median Mortgage:	919
Median Rent:	548
Rent % Household Income:	25.4
Flag Description:	The Maryland flag contains the family crest of the Calvert and Crossland families. Maryland was founded as an English colony in 1634 by Cecil Calvert, the second Lord Baltimore. The black and Gold designs belong to the Calvert family. The red and white design belongs to the Crossland family.

State:	**Massachusetts**
Population:	6016425
Families:	1525198
Households:	2244406
Male %:	48.0%
Female %:	52.0%
Urban %:	84.2%
Average Age:	34.5
HS Diploma %:	80.0%
College Degree %:	34.5%
English Speaking %:	79.0%
Average Commute Time:	28

Figure 6.1: User interface specification for exercise 1



List - StateN...

View Load Snap

Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia

Spotfire - counties of a state (24) - Imported ODBC Data () - [Scatter Plot]

StateID	✓ 24
County	(All)
industry	🟥 ✓ Agricultural Serv
	🟥 ✓ Construction
	🟥 ✓ Finance, Insurar
	🟥 ✓ Manufacturing
	🟪 ✓ Mining
	🟪 ✓ Retail Trade
	🟦 ✓ Services
	🟦 ✓ Transportation a
	🟦 ✓ Wholesale Trad
year	✓ 1993
	✓ 1994
	✓ 1995

CountyID	1705
StateID	24
County	Montgomery Cou
industry	Manufacturing
year	1993
employees	12842
annual payroll ($1000)	289684
establishments	287

(667164, 13485) 25 out of 25 records visible (100.00 %), 0 marked

Table - counties of a state (24)

StateID	County	industry	year	employees	annual payroll ($1000)	stablishments
24	Allegany	Services	1994	8	156856	712
24	Cecil County	Agricultural Services,	1993	1315	4969	131
24	Charles County	Construction	1994	25909	110085	1266
24	Dorchester County	Services	1995	29374	667164	178616
24	Frederick County	Transportation and	1996	5346	143049	536
24	Garrett County	Wholesale Trade	1993	1524	37292	184
24	Harford County	Agricultural Services,	1994	652	9005	136
24	Howard County	Construction	1995	3871	85906	712
24	Kent County	Finance, Insurance,	1996	2464	78068	568
24	Montgomery County	Manufacturing	1993	12842	289684	287
24	Prince George's	Mining	1994	19	0	2
24	Queen Anne's County	Retail Trade	1995	13485	203523	1231
24	Somerset County	Services	1996	21431	487771	1805
24	St. Mary's County	Transportation and	1993	1661	40903	200
24	Talbot County	Wholesale Trade	1994	2994	77196	339
24	Washington County	Agricultural Services,	1995	957	15535	172
24	Wicomico County	Construction	1996	11678	291760	1575
24	Worcester County	Finance, Insurance,	1993	17615	422674	1012

Figure 6.2: User interface specification for exercise 2

Finally, subjects were given the opportunity to freely explore the system, describe problems with the Snap user interface, and offer suggestions for improvement.

The following variables were measured:

- Subjects' background information.
- Learning time.
- Success (y/n or how close to success).
- Time to completion.

This study also observed:

- Cognitive trouble spots (in training and test trials).
- Snap user interface problems.

6.1.2 Results

From the background survey, none of the subjects except the Census programmer had experience with Microsoft Access or SQL, and little exposure to relational database concepts. The Census analysts had significant experience with census data, but generally used flat files or spreadsheets. Each had experience with only basic visualization tools (e.g. Excel charts).

All the subjects completed the training phase in 30-45 minutes. They all were able to complete all three exercises, with occasional help in wading through Access's visual query editor. They accomplished exercise 1 in 2-5 minutes, and exercise 2 in 8-12 minutes. They spent 10-15 minutes on exercise 3 until they were satisfied with their solution.

In general, the subjects were quick to learn the concepts and usage, and were very capable to construct their own coordinated-visualization interfaces. Several stated that they had a sense of satisfaction and power in being able to both (a) so quickly snap powerful exploration environments together, and (b) with just a single click effect exploration across several visualizations and see the many parts operate as a whole. They reported that it made exploration seem effortless, especially in comparison to the standard tools they are used to. As to the subjects' general reaction to Snap, they clearly showed enthusiasm. There may have been social pressure to respond positively, since the subjects knew that the administrator of the experiment was also the developer of the Snap system.

There was an interesting difference between the reaction of the data analysts and programmers (census programmer and computer science students). The programmers commented enthusiastically about the component based programming approach, and the ability to rapidly construct new interfaces. Whereas, the data analysts commented about being able to explore the data thoroughly and efficiently. They did not see it as construction, but as exploration.

In fact, the data analysts performed better than the programmers. They learned the database concepts quicker, completed the exercises quicker, and constructed creative interesting new interfaces. Perhaps they were more motivated by the use of examples involving Census data. Even during the training, they were already trying variations of coordinations and exploring the data. Two pointed out various anomalies in the data. After finishing the exercises, these subjects each voluntarily stayed for an additional hour to discuss and try other examples. All four Census subjects expressed desire to use Snap in their work. In fact, a collaborative effort has been undertaken.

An important result was the creativity and variation evident in the subjects' solutions to exercise 3. Subjects were able to design user interfaces that made cognitive sense to their own perspective on the data. They used a mixture of visualizations including tables, scatter plots, and lists. For example, while the expected design was two scatter plots with a drill-down coordination (one-to-many, select to load), one of the data analyst subjects augmented this design with a pair of lists for the state and county names. The subject stated that this would help to see which state and county was currently selected in the scatter plots, and also allow for accessing states by name which would be difficult with the scatter plot alone. Another subject who preferred to see numeric values placed the counties in a table sorted by number of employees. One had even constructed an interface using the Treemap visualization, which is generally considered a more advanced visualization difficult for novices. In addition to variation in user interfaces, subjects made use of the transitive property of coordination to coordinate visualizations in different pairings.

Overall, subjects did not have problems grasping the cognitive concept of coordinating visualizations. They were able to generate designs by visual duplication and by abstract task description. Results from exercise 3 demonstrated that these users were able to design appropriate coordinated-visualization interfaces. These encouraging results indicate that users can handle a level of design in which they piece together pre-designed components to construct a larger design. Snap apparently finds a middle ground between usage (the realm of end-users) and design (the realm of experienced HCI practitioners) appropriate for these data-savvy users. This validates the primary benefit of Snap, its flexibility.

The problems subjects did have were in manipulating the Snap and Access user interfaces. Creating queries was by far the most difficult part of the construction process for the subjects. Learning to use Access and its query editor is a challenge in such a short time.

6.1.3 User Interface Issues

Understanding the basic Snap model was critical to construction. However, the current Snap user interface and the form fill-in style of the Snap Specification dialog does not reflect this model well. This study identified four major trouble spots in the interface:

1. The terminology of the snap-able actions “select” and “load” caused some confusion. It was not clear enough that these represented user interface actions. Apparently some subjects were confusing “select” with the database query sense of selection.
2. For simplicity, Snap uses the Access query editor. However, this made constructing a drill-down coordination (one-to-many, *select* to *load*) very laborious, and subjects sometimes got lost in the 3 step process: writing the parameterized query, opening the query in a visualization, and specifying the coordination.
3. When constructing interfaces of three or more visualizations, subjects sometimes forgot what coordinations they had constructed between visualizations. They had to recheck each pair.
4. When subjects weren’t quite sure what coordinations they should construct, they would often “just try stuff” and see how it behaves. A snap debugging mode is needed to help them see how the tight-couplings propagate between the visualizations.

Redesigning the Snap user interface around an overview diagram would solve these problems. A node and link diagram could represent the visualizations as nodes and coordinations as links between them. This overview could become the primary user interface for constructing, editing, examining, and debugging coordinations. Such a visual representation with direct-manipulation interaction would closely reflect the conceptual Snap model. Hence, this would likely reduce users’ training time as well.

In addition, while the ability to create queries with Access enables more complex scenarios, it is a burden for common simple coordinations. Basing the Snap Specification dialog on the database schema diagram would more closely match users' mental model of the data. This would simplify constructing drill-down coordinations since Snap could generate the parameterized selection queries automatically. For projection queries, expanding the Snap Menu window to include attribute names would allow users to directly select desired attributes to load into visualizations. Together, these modifications would obviate the need to use Access to manually create queries in common cases. This would further reduce training time to almost nothing.

Also, window management is a serious problem. Subjects spent considerable amounts of time rearranging visualization windows on the screen into nicely tiled layouts. Others have proposed solutions to this general problem (see [KS97] for a review).

6.2 Evaluation of Coordination Operation

The goal of the second study is to measure the added value of coordinated visualizations over independent or single visualizations in terms of user task times and subjective satisfaction for browsing large information spaces. The visual feedback across visualizations could be distracting or disorienting for users. But if there is a benefit, what is its magnitude?

While there are many possibilities, this study examines the overview-and-detail coordination. This coordination has two enhancements over the traditional single-visualization detail-only display:

1. **Overview:** A display enhancement that depicts the full breadth of the data in a compact form, like a table of contents.
2. **Coordination:** An interaction enhancement that allows users to select an item in the overview to scroll the detail to that item. Likewise, directly scrolling the detail highlights the current item in the overview.

Chimera's [CS94] result seems to indicate that overview-and-detail should perform better than detail-only. But, if so, which enhancement is the important factor that causes improved user performance? Is it (a) the information displayed in the overview, or (b) the coordination between the overview and detail?

Hence, the purpose of this study is not to compare a coordinated user interface with the best alternative (see section 2.4 for such studies). Instead, the purpose is to further understand coordination and its users. Specifically, why and how much does the overview-and-detail coordination improve over detail-only, in the context of a single popular type of navigation (one-dimensional scrolling) for browsing tasks? What is the value or detriment of visualizations that are not coordinated? What are users' reactions to these interfaces?

6.2.1 Independent Variables

User interface: A simple textual user interface, constructed with Snap, uses the overview-and-detail coordination for browsing population statistics of 45 of the U.S. states from the Census Bureau's 1990 census. Three treatments: (see Figure 6.1)

1. **Detail-Only:** A single scrolling textual report of the states, in alphabetical order, and their data.

2. **No-Coordination:** The same visualization as Detail-Only, with the addition of a textual overview tiled on the left. The overview displays an alphabetical list of the names of the states. The visualizations are *not* coordinated.
3. **Coordination:** The same visualizations as No-Coordination, with the addition of coordination between them. In Snap, this tightly couples the overview's *select* action to detail's *scroll* action.

At first, the inclusion of the No-Coordination user-interface treatment might seem spurious. However, it is included for two important reasons: First, No-Coordination will reveal which aspect of the coordinated-visualization interface approach is most critical: the multiple visualizations or the coordination. Second, designers actually do build such systems that have uncoordinated visualizations. Microsoft Access is an example. Uncoordination also occurs when using multiple tools by different developers. For example, HCIL members regularly use Spotfire, Excel, Access, and Netscape to examine the HCIL web logs [HS99] and technical-report database. However, they are not coordinated. This is precisely the problem Snap was designed to solve. Hence, it is important to gather data on No-Coordination approaches as well.

Task: A variety of browsing tasks, using a question and answer approach. Nine treatments:

1. **Coverage-yes**: "Does the information include statistics about the state of Ohio?" where Ohio is included in the data.
2. **Coverage-no**: Same as Coverage-Yes, but where the state is not included in the data.
3. **Overview patterns**: "How many states in the list begin with the letter M?"

4. **Visual lookup**: "What is the population of the 6th state from the bottom of the list?"
5. **Nominal lookup**: "What is the population of Georgia?"
6. **Compare-2**: "Which of the following states has higher Median Family Income: California or Washington?"
7. **Compare-5**: "Which of the following 5 states has higher Median Household Income: Florida, Texas, Louisiana, Alaska, or Oregon?"
8. **Search** for target value: "Which state has Average Commute Time of 31?"
9. **Scan** all: "Which state has the highest College Degree %?"

The tasks are listed here in order from easy to difficult based on the experiment results. The actual order they were administered was: 5, 1, 6, 8, 3, 7, 2, 9, 4.

6.2.2 Dependent Variables

User performance time: Time to correctly complete each task, not including reading the task question.

User subjective satisfaction: Subjects rated their satisfaction with each interface on a scale of 1 to 9 on four categories (with scales): comprehensibility (confusing to clear), ease of use (difficult to easy), speed of use (slow to fast), overall satisfaction (terrible to wonderful).

6.2.3 Procedure

The 18 subjects were students and staff from campus, and were paid $10 to participate. A within-subjects design was used. Each subject used all three user interfaces to perform all nine tasks. To avoid repetition, three different but similar sets of task questions were used. To counterbalance for potential order effects, all 6 possible permutations of interface order were each assigned 3 times. The three task sets were not permuted.

For each user interface, subjects were first trained in its use and performed several practice tasks before beginning the timed trials. After finishing all three interface treatments, subjects then completed the subjective satisfaction questionnaire.

6.2.4 Results

Analysis of the data reveals a strong and interesting result. Figure 6.3 shows the mean user-performance times for each task and interface. A 3x9 within-subjects ANOVA reveals that the user interface effect, task effect, and interaction effect are all statistically significant at $p<.001$. Nine one-way ANOVAs reveal that user interface is significant for all 9 tasks at $p<.001$ (see Appendix C section C.2.4 for details of the means, standard deviations, F values and significance levels).

Finally, individual t-tests between each pair of user interfaces within each task determine performance advantages. For tasks 1, 2, and 3, the Coordination and No-Coordination interfaces are both significantly faster than the Detail-Only interface at $p<.001$, but not proven different from each other. Whereas, in tasks 5 through 9, Coordination is significantly faster than both No-Coordination and Detail-Only at $p<.001$, and the latter are not proven different from each other. However, while task 4 (Visual lookup) could be included in the second group of tasks, it may classify as an in-between case. For this task, Coordination is significantly faster than the other two user interfaces at $p<.005$, but No-Coordination is marginally significant over Detail-Only at the $p<.07$ level.

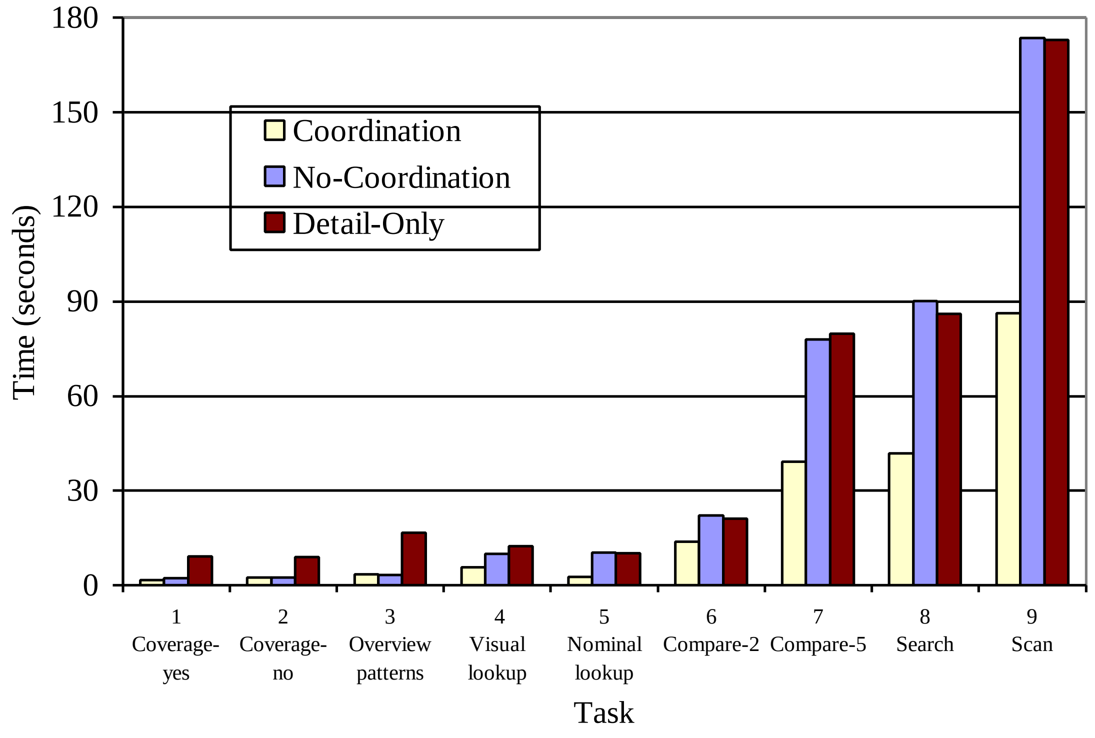


Figure 6.3: Average user performance time for tasks.
The coordinated interface has significantly faster performance in most cases.

First, Coordination results in major improvement in user performance time over Detail-Only for all tasks. On average, Coordination achieves an 80% speedup over Detail-Only for easy tasks and 50% for difficult tasks. The least improvement, about 33%, is in task 6 (compare-2). This task had the lowest interaction-time to thinking-time ratio.

The No-Coordination interface results in a nearly binary pattern, and is likely the source of the interaction effect between task and interface (see Figure 6.4). For tasks 1-3, No-Coordination performs faster than Detail-Only, and its averages are similar to Coordination. In these tasks, subjects only needed the information in the overview to accomplish the task. Whereas, in tasks 5-9 the Coordination interface is faster than No-

Coordination, and the averages for No-Coordination are similar to Detail-Only. In these tasks, subjects needed to access the details of the data. Observing subjects' behavior as they performed these tasks revealed that when using No-Coordination they tended to ignore the overview. The lack of significant difference between No-Coordination and Detail-Only in these cases does not imply that they are necessarily the same. It is conjectured that they are the same due to the observation of the users. In any case, what is important is that Coordination is significantly faster than No-Coordination in these cases. Hence, in tasks where access to details is important, undoubtedly a majority in common applications, coordination is absolutely critical.

	Tasks	
	1-3	4-9
Slower Group	Detail-Only	Detail-Only No-Coordination
Faster Group	No-Coordination Coordination	Coordination

Figure 6.4: User interfaces grouped by user performance in tasks.
The faster groups are significantly faster than the slower groups at $p<0.005$.

Task 4 (Visual lookup) might classify as an in-between case. With No-Coordination, many subjects determined the name of the target state from the overview, then scrolled to it in the detail view. With Detail-Only, they scrolled to the bottom, then scrolled back up while counting, and sometimes lost track. Apparently, this is a case where just having the contextual information of the overview was somewhat advantageous. Even so, Coordination was still a major improvement over both.

In fact, an important result is that Coordination performance times for lookup tasks (4 and 5) are in the same extremely fast range as overview tasks 1-3. Whereas, No-

Coordination times drop to Detail-Only level performance. When looking up details, perhaps the most common task, Coordination especially excels.

In general, overview-and-detail coordination greatly improved performance over detail-only scrolling. Clearly, a major advantage of the coordination is the ability to directly select a target in the overview to immediately locate its details. Whereas, the scrolling interfaces requires careful searching while dragging the scroll bar thumb. Observing the subjects as they performed the tasks revealed that they were more likely to explore when using Coordination. For example, in the Compare-2 and Compare-5 tasks, subjects were more willing to recheck their answers with Coordination. With Detail-Only and No-Coordination subjects spent extra effort to mentally alphabetized the 5 states to compare so as to minimize their scrolling effort. Several subjects reported verbally and on the questionnaire that scrolling was difficult. This is surprising since scrolling is a fundamental component of current GUI systems and perhaps the most common navigational method. The Coordination interface could be considered an improved scroll bar that facilitates exploration.

6.2.5 Subjective Satisfaction

With the satisfaction data (Figure 6.5), a 3x4 within-subjects ANOVA indicates that user interface, subjective satisfaction category, and interaction effect are all significant at p<.001. One-way ANOVAs for each category indicate that Comprehensibility, Ease of use, Speed of use, and Overall Satisfaction are all significant at p<.001 level (see Appendix C section C.2.4 for details of the means, standard deviations, F values and significance levels).

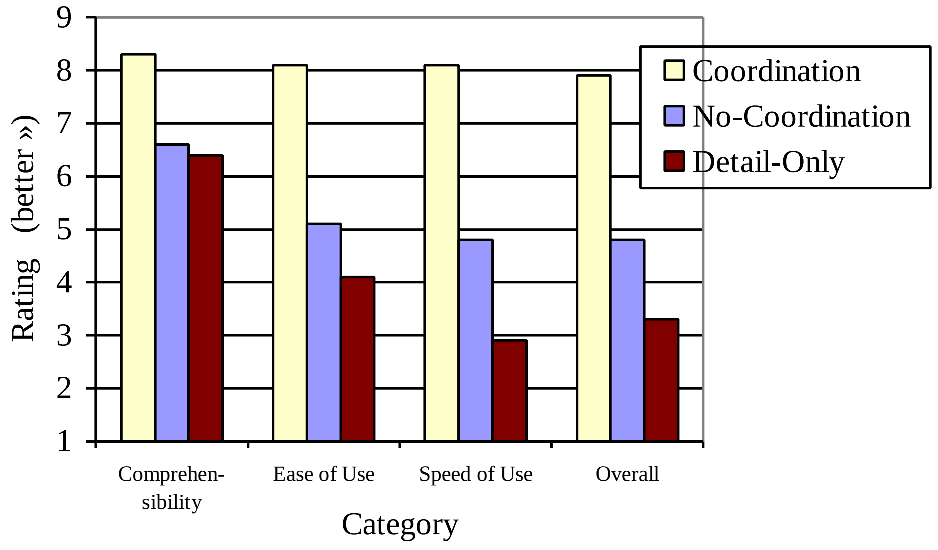


Figure 6.5: Average user subjective satisfaction.
The coordinated interface rates significantly higher in all four categories.

Analyzing each pair of interface treatments within each category reveals that all pairs are significant at $p<.001$ except: Detail-Only and No-Coordination in Ease of Use are significant at $p<.05$ and the same pair in Comprehensibility are not proven different.

Coordination is a clear winner, gaining nearly twice the rankings of Detail-Only and No-Coordination in Ease, Speed, and Overall. On average, subjects ranked No-Coordination 1-2 points higher than Detail-Only, except in Comprehensibility they ranked about the same. While completing the survey, several subjects stated that No-Coordination was only useful for the overview tasks.

6.2.6 Answers

Returning to the research questions: Which factor is more critical, the overview information or the coordination? The answer is nearly binary. If only the overview information is needed, then naturally coordination is not necessary. But for the important cases where access to details is needed, then coordination is everything. What is the magnitude of the benefit? For the three most difficult tasks, the coordinated version cut tasks time in half. This study also reveals the importance of good overview design to enable common questions to be answered directly from the overview.

When first presented with the No-Coordination interface, many subjects immediately attempted to click in the overview expecting the detail view to change, even when they had not yet seen the Coordination interface. Hence, not only were users not distracted by this coordination, but they wanted and expected it! They were visibly distraught when the interface did not behave as they hoped. Even more, they were clearly elated when presented with the Coordination interface, as the subjective satisfaction data indicates. Subjects expressed appreciation for interactive coordination that sped their tasks.

6.3 Combined Analysis

Combining the results from these two studies may indicate the breakpoint at which time savings during coordination operation surpass coordination construction time. In exercise 1 of the first study, subjects constructed the same user interface as was used in the second study for browsing tasks. The time cost of constructing the coordinated interface was about 2-5 minutes, while it saved about 0.6-1.5 minutes over the standard Detail-Only interface for the more difficult tasks. Hence, after just a few tasks, users are already reaping savings when constructing their own coordinated interface. Of course, it is difficult to factor in learning time and effects of sharing saved interfaces. Nevertheless, this simple analysis reveals that customized information visualization is within the grasp of data users.

6.4 Summary

Overall, the overview-and-detail coordination offered a 30-80% speedup over detail-only scrolling for all nine user tasks. While the uncoordinated overview was sufficient for overview only tasks, coordination was critical when accessing details. Users understood and appreciated this coordination.

Data-savvy users successfully and enthusiastically designed and constructed coordinated interfaces of their own. Users showed creativity and variation in their designs. These users are clearly ready for and strongly desire significantly more advanced tools than standard detail-only, uncoordinated, or hard-wired systems. While these cognitive issues were examined within the Snap platform, I believe that these results will apply to similar coordinations and flexibility in other systems.

For practitioners, these studies indicate that Snap can be used in its present form, or that the Snap coordination concepts can be implemented into other systems, to greatly enhance the user experience.

For researchers, several open questions require further study. Other types of coordination, such as brushing and linking, and drill down need to be empirically evaluated. In this study, the use of the No-Coordination user-interface treatment was very successful in identifying the interaction effect between task and coordination. Future studies should exploit this same approach. Also, a browsing task taxonomy is needed for the task independent variable. This study used a variety of exploration tasks, but there may be others to consider. Finally, additional evaluation will be needed to examine the effects of Snap user-interface improvements identified in the study on construction.

Chapter 7: Conclusion

Snap-Together Visualization is a conceptual model, user interface, software architecture, and implemented system that allows data users to rapidly construct customized coordinated-visualization interfaces without programming. Users can dynamically mix and match a variety of visualizations on the fly, and specify common coordinations such as brushing and linking, overview and detail, and drill down. Visualization developers can easily snap-enable their visualizations using a simple API, allowing users to coordinate them with many other visualizations.

Empirical studies of Snap revealed benefits, cognitive issues, and usability concerns. Data-savvy users successfully, enthusiastically, and rapidly designed powerful coordinated-visualization interfaces of their own. An overview-and-detail coordination reliably improved user performance by 30-80% over detail-only and uncoordinated interfaces for most tasks.

7.1 Contributions

This research on Snap-Together Visualization contributes six major innovations:

- **Conceptual model**: a formal model of visualization coordination based on the relational data model and graph model that provides a sound underlying theory and a language for specifying coordinations.
- **User interface**: a user interface for constructing coordinated-visualization interfaces without programming.

- **Software architecture**: an architecture for coordination operation that easily integrates independent visualizations using a simple API, enabling flexibility in data, visualizations, and coordinations.
- **Empirical evaluation**: an evaluation of users' ability to construct and operate their own coordinated-visualization interfaces.
- **Implementation**: an implemented system that realizes the model, user interface, and architecture.
- **Flexibility framework**: a conceptual framework that helps to lay out the space of coordinated-visualization systems based on their level of flexibility in data, visualizations, and coordinations.

Significant evidence validates Snap as both:

- **Useful**: a plethora of examples of Snap usage demonstrate its usefulness and breadth of applicability (Appendix A). At HCIL, snap has been used in several research projects to explore possibilities, and is currently in use at the Census Bureau to expand data visualization capabilities. For example, Fredrikson [FNP99] used Snap to explore approaches for aggregation strategies by temporal, geographical and categorical attributes. Snap has already had significant implementation impact at several organizations including the Census Bureau, Spotfire, WestLaw, and HCIL.
- **Usable**: user studies indicate that Snap is quite usable with training, and user interface improvements have been outlined that will increase its usability and substantially reduce training requirements.

7.2 Uses

7.2.1 Users

Snap can be used for several different purposes. Data users can explore their data by constructing custom visualization user interfaces. User interface designers can quickly prototype many different variations of interfaces, and produce interfaces for data dissemination. Researchers can collaborate by combining their visualizations.

Snap overcomes a serious problem in information visualization research: the isolation of visualizations. Researchers have created a variety of good visualizations, which unfortunately are not coordinated. This makes it difficult for researchers to apply and build on each others work. Snap multiplies the power of visualizations by enabling them in more powerful coordinated-visualization interfaces.

7.2.2 Systems

The Snap model, user interface, and architecture could be employed in a variety of systems. The current Snap implementation focuses on easily enabling the integration of independent visualization tools from the field. To further this goal, Snap and its API could be integrated into a data standard such as ODBC to provide universal support and a closer coupling to data services.

Snap could also be implemented within integrated visualization systems such as Datadesk, Spotfire, Visage, DEVise, Access, and Excel. These systems provide users with a toolbox of cleanly designed visualization components that users could coordinate for exploring data within the system.

Snap could also be used in rapid-application-development (RAD) systems such as Visual Basic. These tools already enable pseudo-programmers to easily manage data schemas and load data into simple visualization components, all using the RAD GUI (e.g. without actually programming). Snap capability would be an ideal next step to enable the users to also coordinate the visualizations with programming.

7.3 Benefits

Snap has many benefits. For visualization researchers and developers, Snap:

- Reuses visualizations. Each visualization needs to be developed only once.
- Simplifies visualization development. Developers can focus efforts on their primary visualization, and use Snap to incorporate supporting visualizations.
- Eliminates the need to program coordinations.
- Steers researchers to more rigorous identification of the purpose and strengths of each visualization. In what situations should a certain visualization be used?
- Provides a platform for studying coordination and its users.
- Provides an API that is useful for other applications too, such as history keeping and collaboration.

For users and interface designers, Snap:

- Provides instant user interfaces for databases, without programming.
- Offers flexibility in data, visualizations, and coordinations, to accommodate varying data, tasks, and users.
- Enables rapid prototyping.
- Offers advantages of coordinated-visualization interfaces, including improved user performance.
- Enables access to many visualizations, and saved groups shared by others.
- Standardizes data format.

- Provides history keeping, data extraction by drag-and-drop, shopping baskets, etc.

7.4 Limitations and Future Work

The limitations and potential future extensions to Snap have been discussed in each of the major chapters. The Snap model focuses on common types of coordinations discovered through experience. These are coordinations for selecting, navigating, and loading data based on discrete data items. Example coordinations include brushing and linking, overview and detail, drill down, synchronized scrolling, and details on demand. Currently, the Snap model is not well suited for attribute-based spatial coordinations of continuous regions. Snap does not yet address other types of coordination such as consistency of dynamic data across visualizations, data mining, or collaboration.

The Snap model could be extended with multiple selection for unions and intersections in drill down coordinations, and could be augmented with attribute-based tight couplings for spatial coordination and data consistency coordination for editing. The Snap user interface could be improved with coordination overview diagrams and the Data Compass to reduce user training and enhance usability. The Snap architecture could be extended with additional coordination controls to increase scalability, the ability to package distributable coordinated-visualization interfaces after construction, and collaboration features. Further evaluation of Snap is needed to study brushing-and-linking and drill-down coordinations, and measure benefits of potential Snap user-interface improvements. In addition, since Snap places significant design capability in the hands of users, guidelines are needed to help them design appropriate coordinated-visualization interfaces for their data.

In the bigger picture, Snap could provide a solution to a rising new problem on the web. Larger databases are increasingly used on the web. With applications such as e-commerce and warehousing, more of the web is becoming data driven. XML is on the rise. Yet, user interfaces on the web are improving slowly. Designers struggle to use frames to provide more advanced coordinated interfaces. Unfortunately, however, the hypertext model is not an appropriate model for coordination. It is uni-directional and embedded in the data. XML provides some relief, since it separates data from presentation, but coordination is missing. Snap can provide the missing link (pun intended). It provides a solid coordination model, and a method for rapidly constructing coordinated interfaces. Visualizations could be simple html and Javascript pages, or more advanced Java applets as Hyperbolic Trees. Web designers could quickly place visualizations into frames and coordinate them. This would solve the primary remaining problem with Snap: distribution to users.

7.5 Conclusions

I believe that Snap-Together Visualization may help information visualization succeed more widely. Snap users can construct the coordinated-visualization interfaces they need for their data and tasks, which would otherwise be difficult and time consuming to obtain.

Yet, this research is only the beginning. Snap opens new possibilities for applied information visualization. It is one step towards ‘crossing the chasm’ [Moo91] – towards helping a wider range of users to explore data, make complex decisions, and apply their creativity [Shn00].

Appendix A: Scenarios

Snap-Together Visualization has been used with a variety of data and visualizations that demonstrate its breadth and usefulness. Example applications include: WestGroup case law, Census Bureau statistics, GIS maps, Maryland State Highway Administration incident data, personal photo libraries, stock market portfolios, web-site logs, mailing address databases, technical-report databases, and hierarchical file structures. These scenarios use a variety of data types, including textual, numeric, geographic, hierarchical, and image. They also employ a variety of visualizations including commercial and non-commercial, and Windows-based as well as web-based. Each scenario includes the specification for the coordinated-visualization interface using the notation of the Snap model from Chapter 3.

A.1 Web-Site Logs

In related work on visualizing web-site logs, Hochheiser [HS99] created scripts to parse web-site log files into an Access database. These files contain data about hits to the HCIL web site. Using Snap, a coordinated-visualization interface (Figure A.1) was easily constructed for examining what other web pages refer many readers to pages on the HCIL web site. The three visualizations at the top (outliner, Treemap, Internet Explorer) form a site browser for the HCIL web site. The outliner and Treemap display the hierarchical structure of the site. Selecting a page in either displays that page in IE.

The Treemap shows that the HCIL home page, Pad++, and the Visible Human Explorer are the most frequently visited pages.

The two visualizations at the bottom (scatter plot, and IE) display other pages that refer readers to the selected page in the site browser. The plot shows referring pages along the X-axis and the number of hits referred (during October 1998) on the Y-axis. Selecting the most frequent referrer (110 hits) to the HCIL home page reveals Human Factors International in IE. Exploring reveals other common referrers, including Ben Shneiderman's page, the Department page, and Yahoo's HCI institutes page. Selecting the Visible Human Explorer page in the outliner shows nearly 1000 hits from the National Library of Medicine page. Selecting to open this page indeed reveals a prominent link to the HCIL page. Naturally, HCIL lab members explored to discover referrer patterns to their personal pages.

The Snap specification for this interface is:

Visualizations = { (outliner, pages), (Treemap, pages), (IE_{top}, pages),
(plot, pageReferrers), (IE_{bottom}, pageReferrers) }

Coordinations = { ((outliner, select), (Treemap, select)),
((outliner, select), (IE_{top}, load-PK)),
((outliner, select), (plot, load-FK_{page})),
((plot, select), (IE_{bottom}, load-PK)) }

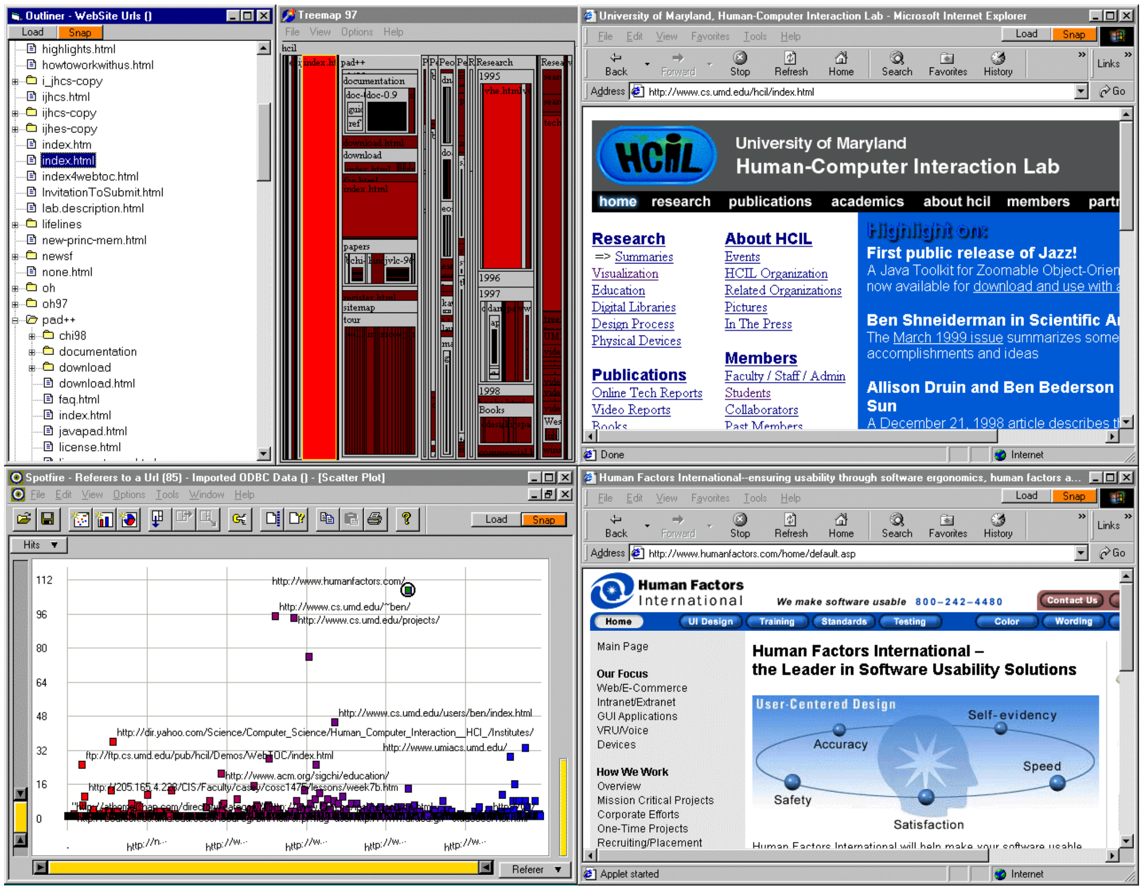

Outliner - WebSite Urls ()
Treemap 97
University of Maryland, Human-Computer Interaction Lab - Microsoft Internet Explorer
Spotfire - Referers to a Url (85) - Imported ODBC Data () - [Scatter Plot]
Human Factors International--ensuring usability through software ergonomics, human factors a...
Load
Snap
Hits
Referer

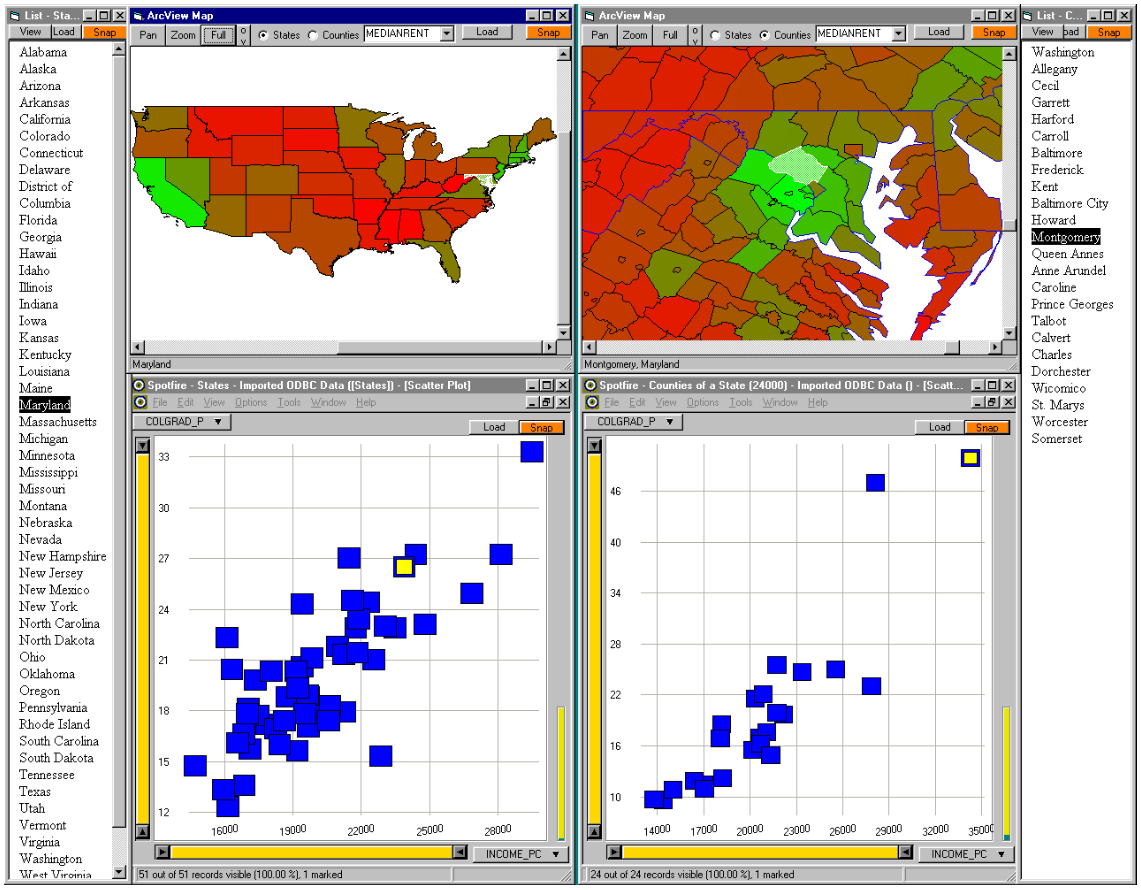
Figure A.1: Web-site logs scenario

A.2 Census Data

Figure A.2 is an interface for exploring Census population data of U.S. states (left) and counties (right). Users can explore from nominal, geographic and numeric perspectives. Selecting Maryland reveals that it ranks very high in terms of income per capita and percent college graduates. Maryland has two counties that have much higher percentage of college graduates that the others. One of these, Montgomery County, has the highest per capita income and is clearly located just north of DC.

This example demonstrates the use of ESRI MapObjects, a component of the popular ArcView GIS software.

Snap is in use at the Census Bureau to prototype user interfaces for CD-ROM products. Census analysts have also found the capability to relate data between maps and plots extremely helpful. Continued work on Snap at Census has already enabled multiple selection for brushing and linking, and connection to intranet-based Oracle database servers.



51 out of 51 records visible (100.00 %), 1 marked

24 out of 24 records visible (100.00 %), 1 marked

Figure A.2: Census data scenario

The Snap specification for this interface is:

$$\text{Visualizations} = \{\, (\text{map}_{\text{states}}, \text{states}), (\text{plot}_{\text{states}}, \text{states}), (\text{list}_{\text{states}}, \text{states}), (\text{map}_{\text{counties}}, \text{counties}), (\text{plot}_{\text{counties}}, \text{counties}), (\text{list}_{\text{counties}}, \text{counties}) \,\}$$

Coordinations = { ((map$_{\text{states}}$, select), (plot$_{\text{states}}$, select)),
((map$_{\text{states}}$, select), (list$_{\text{states}}$, select)),
((map$_{\text{counties}}$, select), (plot$_{\text{counties}}$, select)),
((map$_{\text{counties}}$, select), (list$_{\text{counties}}$, select)),
((map$_{\text{states}}$, select), (map$_{\text{counties}}$, zoom)),
((map$_{\text{states}}$, select), (plot$_{\text{counties}}$, load-FK$_{\text{state}}$)),
((plot$_{\text{counties}}$, load-FK$_{\text{state}}$), (list$_{\text{counties}}$, load-FK$_{\text{state}}$)) }

A.3 Photo Libraries

Snap was used in an HCIL research project on user interfaces for personal digital-photo libraries [KTS00] to explore many possible designs. The lab has accumulated a database of scanned photos of lab members and activities spanning 10 years. It includes annotations such as members' names, dates, locations, and other information.

In Figure A.3, a thumbnail visualization shows a collection of a few hundred photos. The scatter plot displays a time-line overview of the photos, with date on the X-axis and members' names on the Y. Uses can see trends and patterns. For example, vertical stripes of dots represent group events, pictures of many members on the same date. The large stripe in the middle is many photos from the 1992 HCIL Open House. Selecting a photo from winter '89 displays the full-size photo from a ski trip, a list of names of members in the photo, and details of photo attributes.

Other interface variations include locating photos by members' names or locations, selecting a person in a photo to find other pictures of that person, etc.

The Snap specification for this interface is:

Visualizations = { (thumbnails, photos), (plot, photos), (IE, photos),
(list$_{\text{people}}$, appearances), (list$_{\text{details}}$, photos) }

Coordinations = { ((thumbnails, select), (plot, select)),
((thumbnails, select), (IE, load-PK)),
((thumbnails, select), (list$_{\text{people}}$, load-FK$_{\text{photo}}$)),
((thumbnails, select), (list$_{\text{details}}$, load-PK)) }

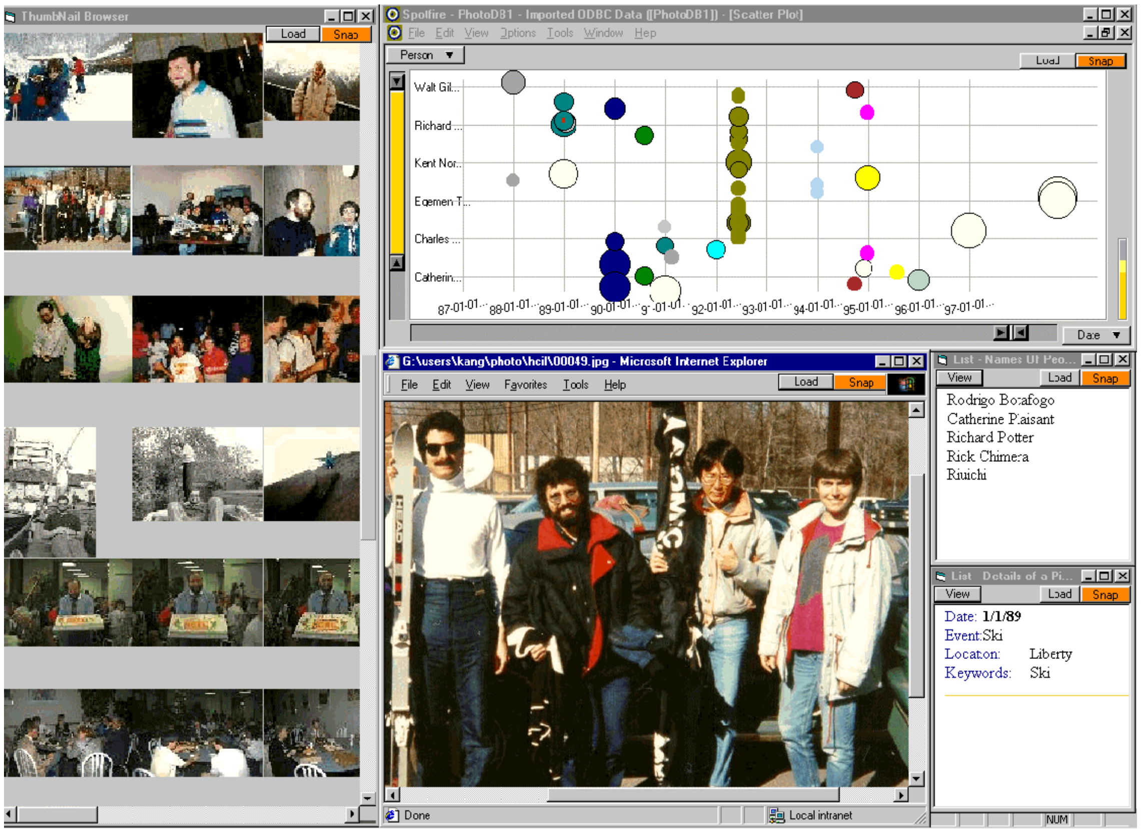


Figure A.3: Photo libraries scenario

A.4 WestLaw Case-Law Documents

Significant inspiration for the Snap concept resulted from an HCIL research project for WestLaw on visualization of case-law documents. Snap was used to prototype work-benches for legal analysts. A major task that the analysts perform is to search large case-law document databases using keywords, and then examine resulting cases for relevance to a current case.

This user interface in Figure A.4 is for browsing search results. The visualizations at the top of the screen display the hits resulting from a search. The Snap search window is used to enter search terms (described in Chapter 4). The hits are displayed both textually and graphically by date and search relevance. Selecting a case displays it in the case viewer at the bottom of the screen.

The case viewer displays the text-intensive details of the case in a manner that supports rapid navigation. A case is composed of a judge's decision text, which is partitioned into sections. Each section has a WestLaw headnote, containing a categorization and annotation. WestLaw's existing user interface simply listed 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-frame synchronized-scrolling approach is more appropriate. The main list visualization on the right displays the text of the case by sections. The center list displays WestLaw headnotes for each section, and synchronizes scrolling with the main text. Since many cases are long, containing 10 to 50 headnotes, users can quickly jump to a section by selecting section numbers from the overview list on the left.

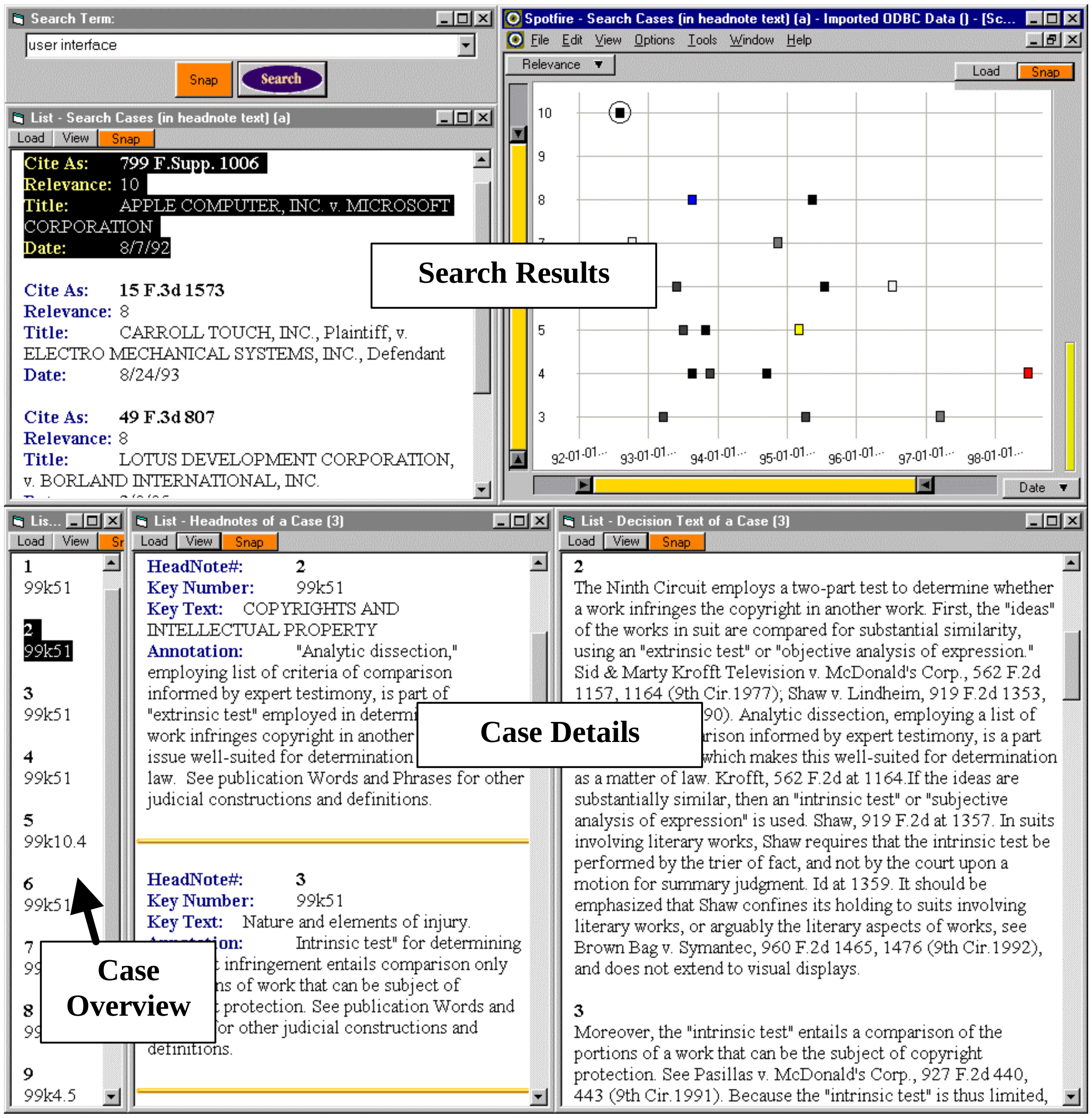
This example demonstrates how Snap would be ideal for rapid web-based user interface construction. In fact, based on this prototype, WestLaw did implement this case-viewer design in their web site (www.westlaw.com).

The Snap specification for this interface is:

Visualizations = { (search, phrases), ($\text{list}_{\text{hits}}$, cases), (plot, cases),
($\text{list}_{\text{overview}}$, sections), ($\text{list}_{\text{headnotes}}$, sections),
($\text{list}_{\text{text}}$, sections) }

Coordinations = { ((search, search), ($\text{list}_{\text{hits}}$, $\text{load-FK}_{\text{search}}$)),
(($\text{list}_{\text{hits}}$, $\text{load-FK}_{\text{search}}$), (plot, $\text{load-FK}_{\text{search}}$)),
(($\text{list}_{\text{hits}}$, select), (plot, select)),
(($\text{list}_{\text{hits}}$, select), ($\text{list}_{\text{overview}}$, $\text{load-FK}_{\text{case}}$)),

$((\text{list}_{\text{overview}}, \text{load-FK}_{\text{case}}), (\text{list}_{\text{headnotes}}, \text{load-FK}_{\text{case}})),$
$((\text{list}_{\text{overview}}, \text{load-FK}_{\text{case}}), (\text{list}_{\text{text}}, \text{load-FK}_{\text{case}}))$
$((\text{list}_{\text{overview}}, \text{select}), (\text{list}_{\text{headnotes}}, \text{scroll})),$
$((\text{list}_{\text{headnotes}}, \text{scroll}), (\text{list}_{\text{text}}, \text{scroll})), \quad \}$



Search Term:
user interface
Snap
Search
Spotfire - Search Cases (in headnote text) (a) - Imported ODBC Data () - [Sc...
File Edit View Options Tools Window Help
Relevance
Load
Snap
Search Results
Date
List - Search Cases (in headnote text) (a)
Load View Snap
Lis...
List - Headnotes of a Case (3)
List - Decision Text of a Case (3)
Case Details
Case Overview

Cite As: 799 F.Supp. 1006
Relevance: 10
Title: APPLE COMPUTER, INC. v. MICROSOFT CORPORATION
Date: 8/7/92

Cite As: 15 F.3d 1573
Relevance: 8
Title: CARROLL TOUCH, INC., Plaintiff, v. ELECTRO MECHANICAL SYSTEMS, INC., Defendant
Date: 8/24/93

Cite As: 49 F.3d 807
Relevance: 8
Title: LOTUS DEVELOPMENT CORPORATION, v. BORLAND INTERNATIONAL, INC.

1 99k51
2 99k51
3 99k51
4 99k51
5 99k10.4
6 99k51
7 99
8 99
9 99k4.5

HeadNote#: 2
Key Number: 99k51
Key Text: COPYRIGHTS AND INTELLECTUAL PROPERTY
Annotation: "Analytic dissection," employing list of criteria of comparison informed by expert testimony, is part of "extrinsic test" employed in determi work infringes copyright in another issue well-suited for determination law. See publication Words and Phrases for other judicial constructions and definitions.

HeadNote#: 3
Key Number: 99k51
Key Text: Nature and elements of injury.
ion: Intrinsic test" for determining infringement entails comparison only ns of work that can be subject of protection. See publication Words and or other judicial constructions and definitions.

2
The Ninth Circuit employs a two-part test to determine whether a work infringes the copyright in another work. First, the "ideas" of the works in suit are compared for substantial similarity, using an "extrinsic test" or "objective analysis of expression." Sid & Marty Krofft Television v. McDonald's Corp., 562 F.2d 1157, 1164 (9th Cir.1977); Shaw v. Lindheim, 919 F.2d 1353, 90). Analytic dissection, employing a list of rison informed by expert testimony, is a part which makes this well-suited for determination as a matter of law. Krofft, 562 F.2d at 1164.If the ideas are substantially similar, then an "intrinsic test" or "subjective analysis of expression" is used. Shaw, 919 F.2d at 1357. In suits involving literary works, Shaw requires that the intrinsic test be performed by the trier of fact, and not by the court upon a motion for summary judgment. Id at 1359. It should be emphasized that Shaw confines its holding to suits involving literary works, or arguably the literary aspects of works, see Brown Bag v. Symantec, 960 F.2d 1465, 1476 (9th Cir.1992), and does not extend to visual displays.

3
Moreover, the "intrinsic test" entails a comparison of the portions of a work that can be the subject of copyright protection. See Pasillas v. McDonald's Corp., 927 F.2d 440, 443 (9th Cir.1991). Because the "intrinsic test" is thus limited,

Figure A.4: Case-law scenario

A.5 Highway Incident Data

For an HCIL research project on visual aggregation strategies, Fredrikson [FNP99] used Snap with Maryland State Highway Administration incident data. She identified temporal, geographical and categorical attributes as ideal candidates for aggregation. The drill-down coordination was used to allow users to select aggregates in one visualization to display aggregate contents in another visualization. For example, Figure A.5 displays aggregations of highway accidents by day of the week in a bar chart. Selecting Monday, which had the most accidents, reveals the locations of individual accidents on the road map of the Baltimore, MD area.

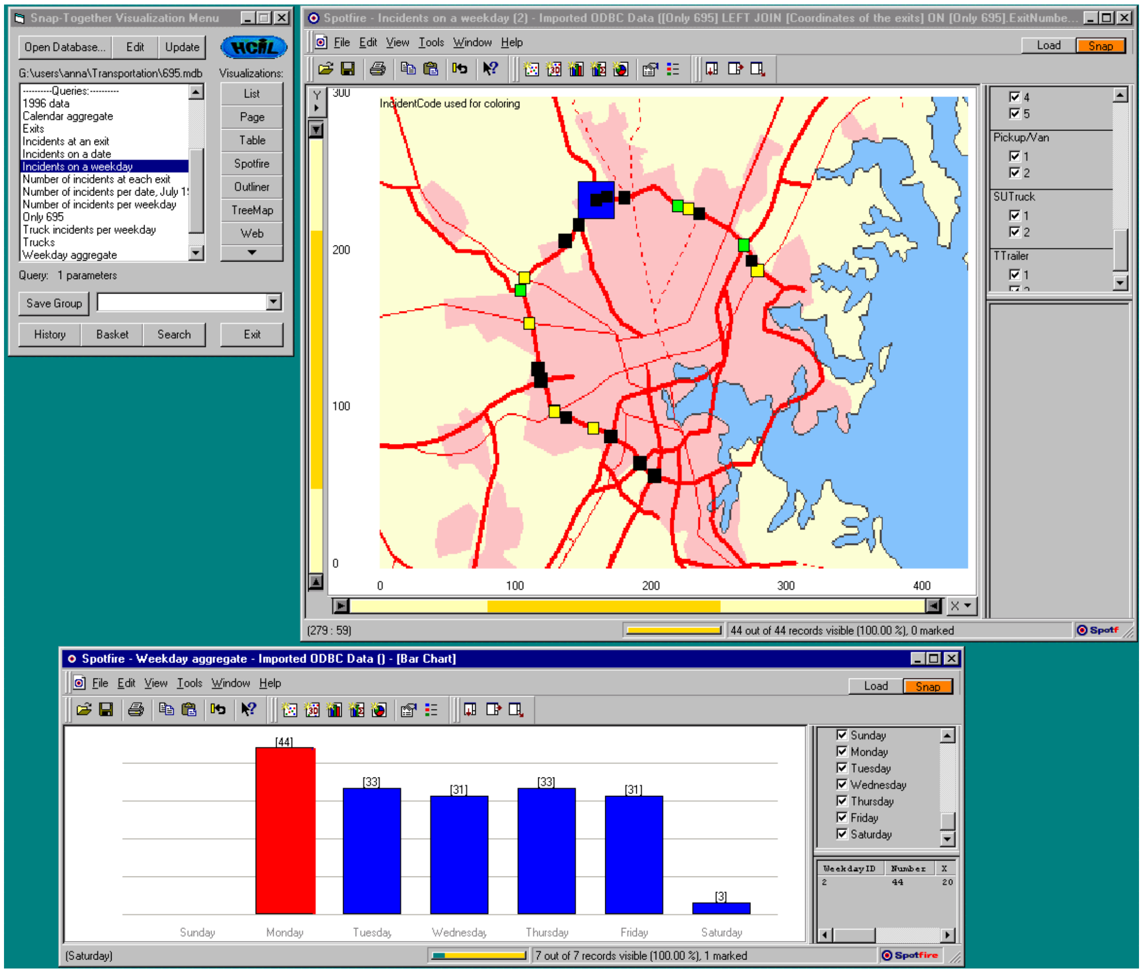

Snap-Together Visualization Menu
Open Database...
Edit
Update
G:\users\anna\Transportation\695.mdb
Visualizations:
---------Queries:---------
1996 data
Calendar aggregate
Exits
Incidents at an exit
Incidents on a date
Incidents on a weekday
Number of incidents at each exit
Number of incidents per date, July 1
Number of incidents per weekday
Only 695
Truck incidents per weekday
Trucks
Weekday aggregate
List
Page
Table
Spotfire
Outliner
TreeMap
Web
Query: 1 parameters
Save Group
History
Basket
Search
Exit
Spotfire - Incidents on a weekday (2) - Imported ODBC Data ([Only 695] LEFT JOIN [Coordinates of the exits] ON [Only 695].ExitNumbe...
File Edit View Tools Window Help
Load
Snap
IncidentCode used for coloring
300
200
100
0
0
100
200
300
400
4
5
Pickup/Van
1
2
SUTruck
1
2
TTrailer
1
(279 : 59)
44 out of 44 records visible (100.00 %), 0 marked
Spotfire - Weekday aggregate - Imported ODBC Data () - [Bar Chart]
[44]
[33]
[31]
[33]
[31]
[3]
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
WeekdayID Number %
2 44 20
(Saturday)
7 out of 7 records visible (100.00 %), 1 marked

Figure A.5: Highway incident data scenario

While the data set used in this project was not large (~1000 tuples), this technique demonstrates how Snap can be used to explore very large-scale relations using drill down. For example, 1,000,000 traffic incidents could be aggregated into 1,000 aggregates, each with 1,000 incidents. This could be displayed with two coordinated visualizations, an overview of 1,000 points, and a detail view of 1,000. Furthermore, this approach can be repeated by chaining several visualizations, adding an additional visualization for each level to multiply by powers of 1,000.

The Snap specification for this interface is:

Visualizations = { (barchart, dayAggregates), (map, incidents) }

Coordinations = { ((barchart, select), (map, load-FK_{day})) }

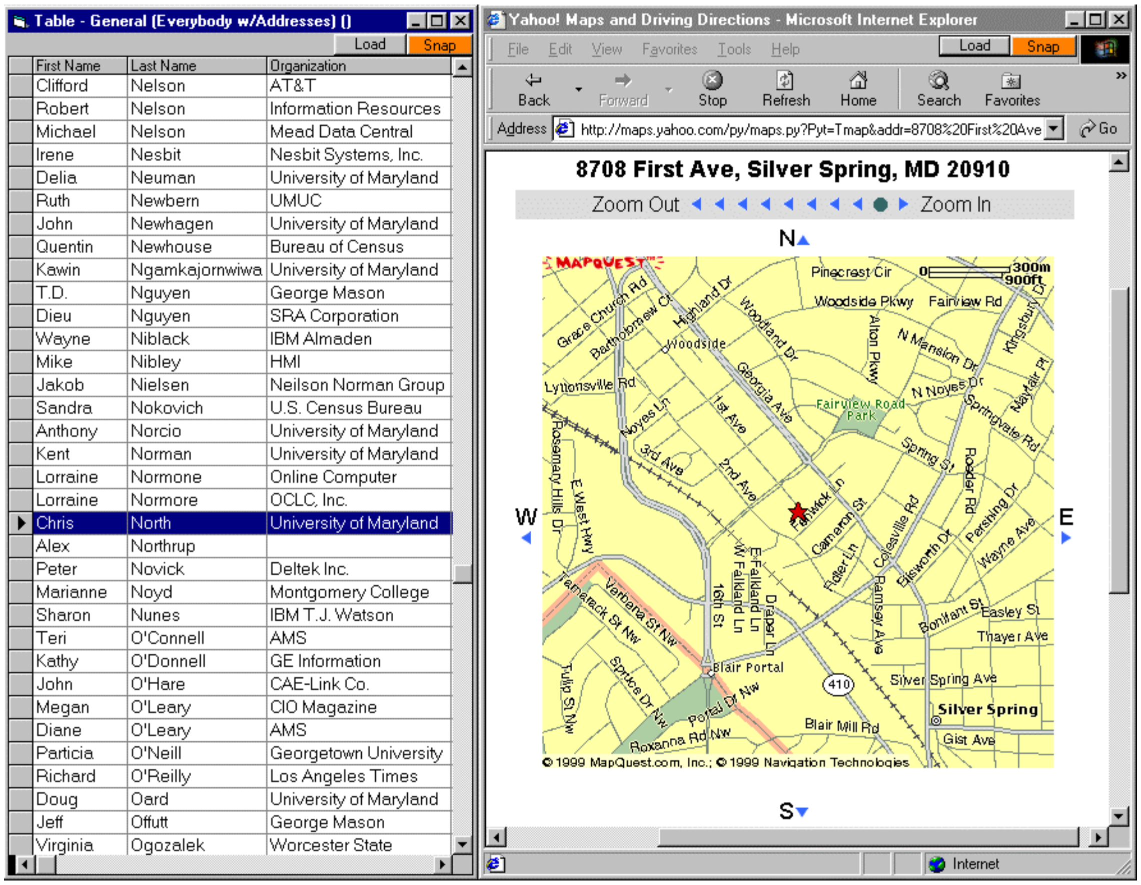
A.6 Mailing Address Database

In Figure A.6, Snap is used to explore addresses in the HCIL mailing-list database. The names are displayed in a simple table. The table is coordinated to IE, in which a query is loaded that formulates a mailing address as a URL query string to Yahoo Maps. Then, selecting a name in the mailing list displays a map of the location of that address. This example demonstrates how web services such as Yahoo Maps can be used as snap-able visualizations.

The Snap specification for this interface is:

Visualizations = { (table, addresses), (IE, addressQueryStrings) }

Coordinations = { ((table, select), (IE, load-PK))}



http://maps.yahoo.com/py/maps.py?Pyt=Tmap&addr=8708%20First%20Ave

8708 First Ave, Silver Spring, MD 20910

First Name	Last Name	Organization
Clifford	Nelson	AT&T
Robert	Nelson	Information Resources
Michael	Nelson	Mead Data Central
Irene	Nesbit	Nesbit Systems, Inc.
Delia	Neuman	University of Maryland
Ruth	Newbern	UMUC
John	Newhagen	University of Maryland
Quentin	Newhouse	Bureau of Census
Kawin	Ngamkajornwiwa	University of Maryland
T.D.	Nguyen	George Mason
Dieu	Nguyen	SRA Corporation
Wayne	Niblack	IBM Almaden
Mike	Nibley	HMI
Jakob	Nielsen	Neilson Norman Group
Sandra	Nokovich	U.S. Census Bureau
Anthony	Norcio	University of Maryland
Kent	Norman	University of Maryland
Lorraine	Normore	Online Computer
Lorraine	Normore	OCLC, Inc.
Chris	North	University of Maryland
Alex	Northrup	
Peter	Novick	Deltek Inc.
Marianne	Noyd	Montgomery College
Sharon	Nunes	IBM T.J. Watson
Teri	O'Connell	AMS
Kathy	O'Donnell	GE Information
John	O'Hare	CAE-Link Co.
Megan	O'Leary	CIO Magazine
Diane	O'Leary	AMS
Particia	O'Neill	Georgetown University
Richard	O'Reilly	Los Angeles Times
Doug	Oard	University of Maryland
Jeff	Offutt	George Mason
Virginia	Ogozalek	Worcester State

Figure A.6: Mailing address database scenario

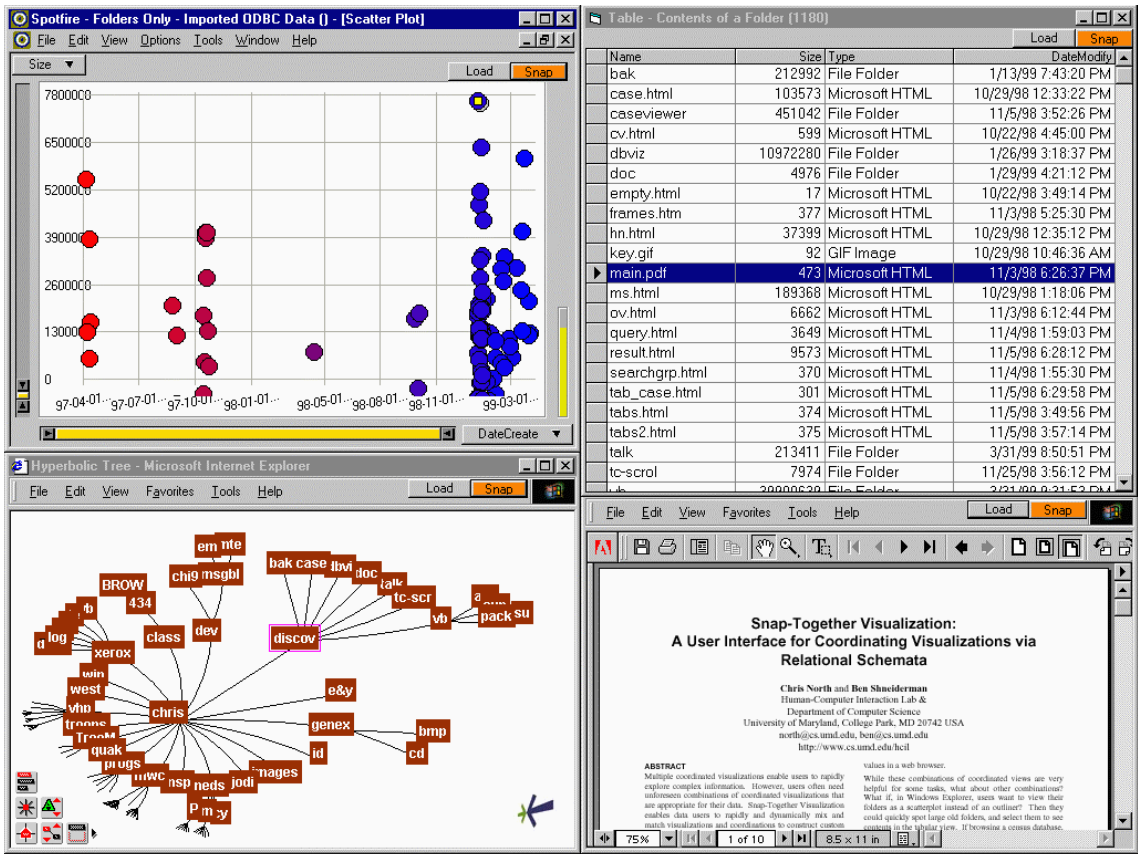
A.7 Files and Folders

The file-folders scenario in Chapter 1 demonstrates how Java applets such as Hyperbolic Trees can be snap-enabled using IE (Figure A.7). IE is also useful as a general-purpose file viewer for images, HTML, PDF and Word documents, etc.

The Snap specification for this interface is:

Visualizations = { (plot, folders), (hyperbolic, folders), (table, files), (IE, files) }

Coordinations = { ((plot, select), (hyperbolic, select)),
((plot, select), (table, load-FK$_{folder}$)),
((table, select), (IE, Load-PK)) }


Spotfire - Folders Only - Imported ODBC Data () - [Scatter Plot]
Size
Load
Snap
DateCreate
Hyperbolic Tree - Microsoft Internet Explorer
Table - Contents of a Folder (1180)
File Edit View Favorites Tools Help

Name	Size	Type	DateModify
bak	212992	File Folder	1/13/99 7:43:20 PM
case.html	103573	Microsoft HTML	10/29/98 12:33:22 PM
caseviewer	451042	File Folder	11/5/98 3:52:26 PM
cv.html	599	Microsoft HTML	10/22/98 4:45:00 PM
dbviz	10972280	File Folder	1/26/99 3:18:37 PM
doc	4976	File Folder	1/29/99 4:21:12 PM
empty.html	17	Microsoft HTML	10/22/98 3:49:14 PM
frames.htm	377	Microsoft HTML	11/3/98 5:25:30 PM
hn.html	37399	Microsoft HTML	10/29/98 12:35:12 PM
key.gif	92	GIF Image	10/29/98 10:46:36 AM
main.pdf	473	Microsoft HTML	11/3/98 6:26:37 PM
ms.html	189368	Microsoft HTML	10/29/98 1:18:06 PM
ov.html	6662	Microsoft HTML	11/3/98 6:12:44 PM
query.html	3649	Microsoft HTML	11/4/98 1:59:03 PM
result.html	9573	Microsoft HTML	11/5/98 6:28:12 PM
searchgrp.html	370	Microsoft HTML	11/4/98 1:55:30 PM
tab_case.html	301	Microsoft HTML	11/5/98 6:29:58 PM
tabs.html	374	Microsoft HTML	11/5/98 3:49:56 PM
tabs2.html	375	Microsoft HTML	11/5/98 3:57:14 PM
talk	213411	File Folder	3/31/99 8:50:51 PM
tc-scrol	7974	File Folder	11/25/98 3:56:12 PM

Snap-Together Visualization:
A User Interface for Coordinating Visualizations via Relational Schemata

Chris North and Ben Shneiderman
Human-Computer Interaction Lab &
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
Multiple coordinated visualizations enable users to rapidly explore complex information. However, users often need unforeseen combinations of coordinated visualizations that are appropriate for their data. Snap-Together Visualization enables data users to rapidly and dynamically mix and match visualizations and coordinations to construct custom

values in a web browser.

While these combinations of coordinated views are very helpful for some tasks, what about other combinations? What if, in Windows Explorer, users want to view their folders as a scatterplot instead of an outliner? Then they could quickly spot large old folders, and select them to see contents in the tabular view. If browsing a census database,

Figure A.7: Files and folders scenario

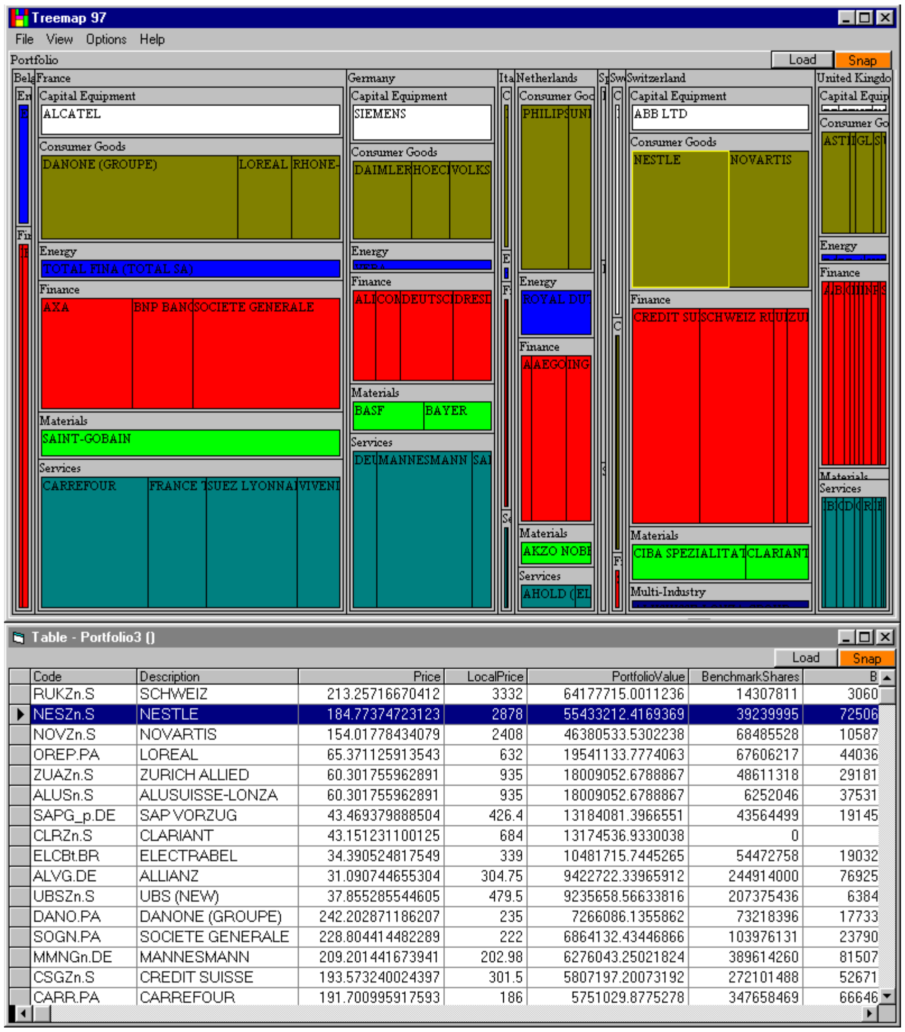
A.8 Stock Market Portfolios

In data analysis, it is often useful to view both the graphical visualization as well as the detailed numeric spreadsheet. In Figure A.8, Snap is used to display a financial stock portfolio. Brushing and linking relate the Treemap and spreadsheet.

The Snap specification for this interface is:

Visualizations = { (treemap, stocks), (table, stocks) }

Coordinations = { ((treemap, select), (table, select)) }



Code	Description	Price	LocalPrice	PortfolioValue	BenchmarkShares	B
RUKZn.S	SCHWEIZ	213.25716670412	3332	64177715.0011236	14307811	3060
NESZn.S	NESTLE	184.77374723123	2878	55433212.4169369	39239995	72506
NOVZn.S	NOVARTIS	154.01778434079	2408	46380533.5302238	68485528	10587
OREP.PA	LOREAL	65.371125913543	632	19541133.7774063	67606217	44036
ZUAZn.S	ZURICH ALLIED	60.301755962891	935	18009052.6788867	48611318	29181
ALUSn.S	ALUSUISSE-LONZA	60.301755962891	935	18009052.6788867	6252046	37531
SAPG_p.DE	SAP VORZUG	43.469379888504	426.4	13184081.3966551	43564499	19145
CLRZn.S	CLARIANT	43.151231100125	684	13174536.9330038	0	
ELCBt.BR	ELECTRABEL	34.390524817549	339	10481715.7445265	54472758	19032
ALVG.DE	ALLIANZ	31.090744655304	304.75	9422722.33965912	244914000	76925
UBSZn.S	UBS (NEW)	37.855285544605	479.5	9235658.56633816	207375436	6384
DANO.PA	DANONE (GROUPE)	242.202871186207	235	7266086.1355862	73218396	17733
SOGN.PA	SOCIETE GENERALE	228.804414482289	222	6864132.43446866	103976131	23790
MMNGn.DE	MANNESMANN	209.201441673941	202.98	6276043.25021824	389614260	81507
CSGZn.S	CREDIT SUISSE	193.573240024397	301.5	5807197.20073192	272101488	52671
CARR.PA	CARREFOUR	191.700995917593	186	5751029.8775278	347658469	66646

Figure A.8: Stock market portfolio scenario

A.9 Visible Human Images

As described in Chapter 3, Snap could also be used in medical and scientific domains to relate physical structures in images to other types of information. The mockup in Figure A.9 demonstrates the concept. While this example has not been implemented, it can be done with Snap. For example, html image maps in IE have been used with Snap. An image map of the U.S. was used prior to the use of ArcView

(Figure A.10). Another approach might be to use a volume visualization tool that supports the selection of structural objects.

The Snap specification for this interface would be:

Visualizations = { (volumeviz, structures), (outliner, structures) }

Coordinations = { ((volumeviz, select), (outliner, select)) }

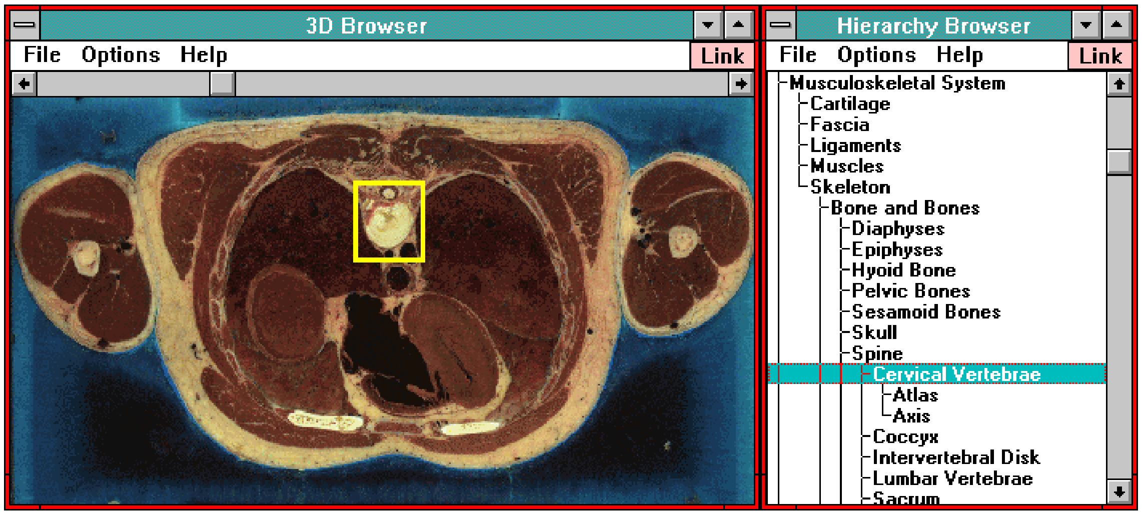


Figure A.9: Visible Human images scenario

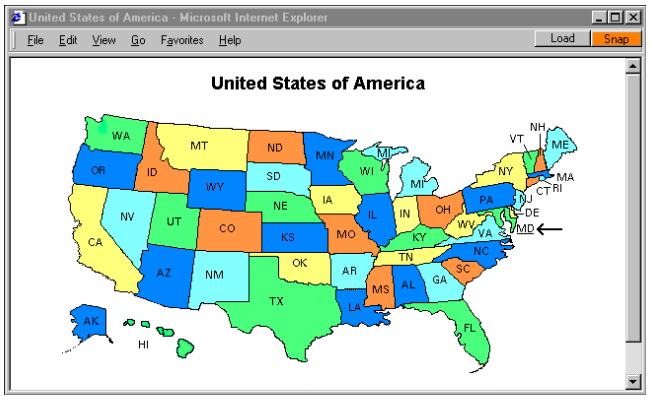


Figure A.10: Image map in IE

A.10 Summary

These scenarios also demonstrate the serious need that Snap fulfills. Without the use of Snap, scenarios such as the web logs example simply could not be readily accomplished. They would require significant custom programming, or the difficult and tedious use of uncoordinated displays.

These examples demonstrate how Snap has already been highly applicable and useful in many projects. It has been useful to both researchers and practitioners, and has already had an impact at several organizations including HCIL, the Census Bureau, Spotfire and WestLaw.

Appendix B: Review of Coordinated-Visualization Systems

Coordinated visualization systems have become an important and diverse topic. Many such systems have been built. Most of these systems are data flexible (defined in Chapter 2). That is, typically they can be used to visualize different data sets, but are usually fixed in terms of the visualizations and coordinations in their user interface. This Appendix reviews many of these systems from the field. As in the rest of this dissertation, the focus is on coordinations for information exploration.

A simple taxonomy is used to lay out the space of these systems [NS97], loosely based on the conceptual model of visualization coordination described in Chapter 3. Visualizations have two basic classes of actions:

- **Select**: Users can select and highlight data items in the visualization to express interest in them, or possibly to initiate other forms of manipulation on them.
- **Navigate**: Users can navigate the visualization to focus on data items or to display other data items (e.g. scroll, pan, zoom, slice, rotate, ascend/descend tree, follow link, open file, etc.). For the purposes of this taxonomy, navigate also includes the *load* action to load other data into a visualization as a form of navigation through the larger data context.

Coordinating a pair of visualizations tightly couples one of these actions in the one visualization to another action in the other visualization. The taxonomy classifies coordinations by the three possible combinations of actions (Figure B.1):

1. Select ↔ select
2. Navigate ↔ navigate
3. Select ↔ navigate (which is equivalent to navigate ↔ select due to bi-directionality)

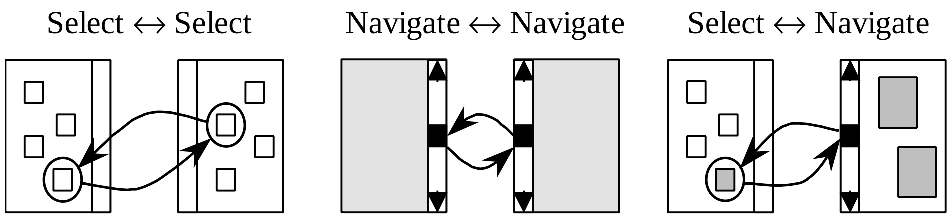


Figure B.1: A taxonomy of coordinations

B.1 Select ↔ Select

This coordination tightly couples selecting items in one visualization to selecting items in another visualization, to help users correlate equivalent or related items. When users select (highlight, paint, brush) an item (or set of items) in one visualization, the system immediately highlights the equivalent item (or set), representing the same underlying data elements, in the other visualization.

Many exploratory data analysis systems use this coordination to visualize high-dimensional data point sets with multiple coordinated plots. Common examples are Datadesk [Vel88], SAS Insight, JMP, EDV [EW95], Spotfire [AW95], XGobi [BCS96], XmdvTool [WA95]. Invention of this brushing-and-linking concept is generally credited to Prim-9 [FFT74] or Newton [New78]. [Mon89] introduced brushing to GIS by brushing between plots and geographic choropleth maps. XmdvTool provides the capability to brush regions in attribute space as well as individual data items. For example, in Figure B.2 an n-dimensional region is selected in both the plot matrix and parallel-coordinates graph.

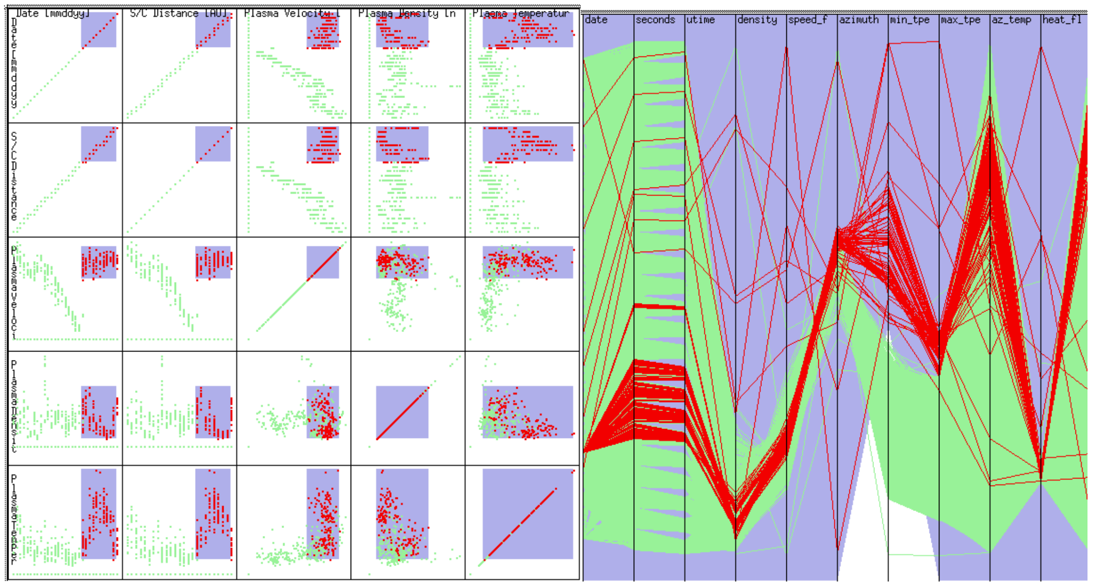


Figure B.2: XmdvTool

For examples with other types of data, the Navigational View Builder [MFH95] (Figure B.3) brushes nodes in hierarchical information, linking Treemaps (emphasizing numerical and categorical attributes), ConeTrees (emphasizing structure), and outliners (emphasizing node names). With Lilac [Bro91], a two-window document editor, selecting text in the WYSIWYG page window also selects the corresponding text in the source text window (similar to HTML code).

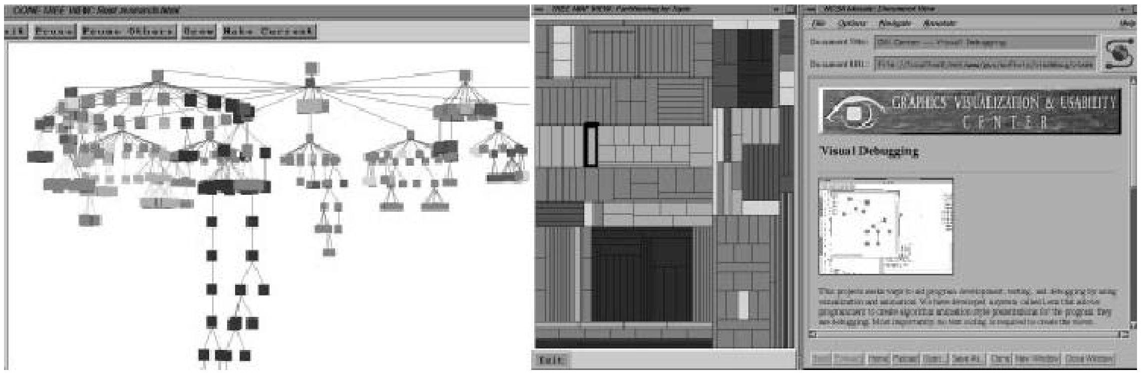


Figure B.3: Navigational View Builder

An interesting variation is the Attribute Explorer [STD95], which uses additive encoding of multiple brushes (Figure B.4). It displays multi-dimensional data in a series of 1-dimensional histograms, and users can select a range in each histogram. Then, data points are color coded by the number of attribute selections they are contained in. Points that satisfy more selections are lighter, fewer selections are darker.

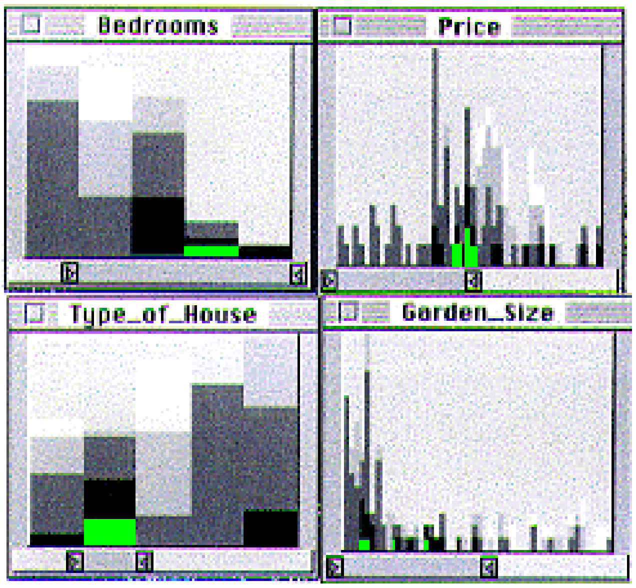
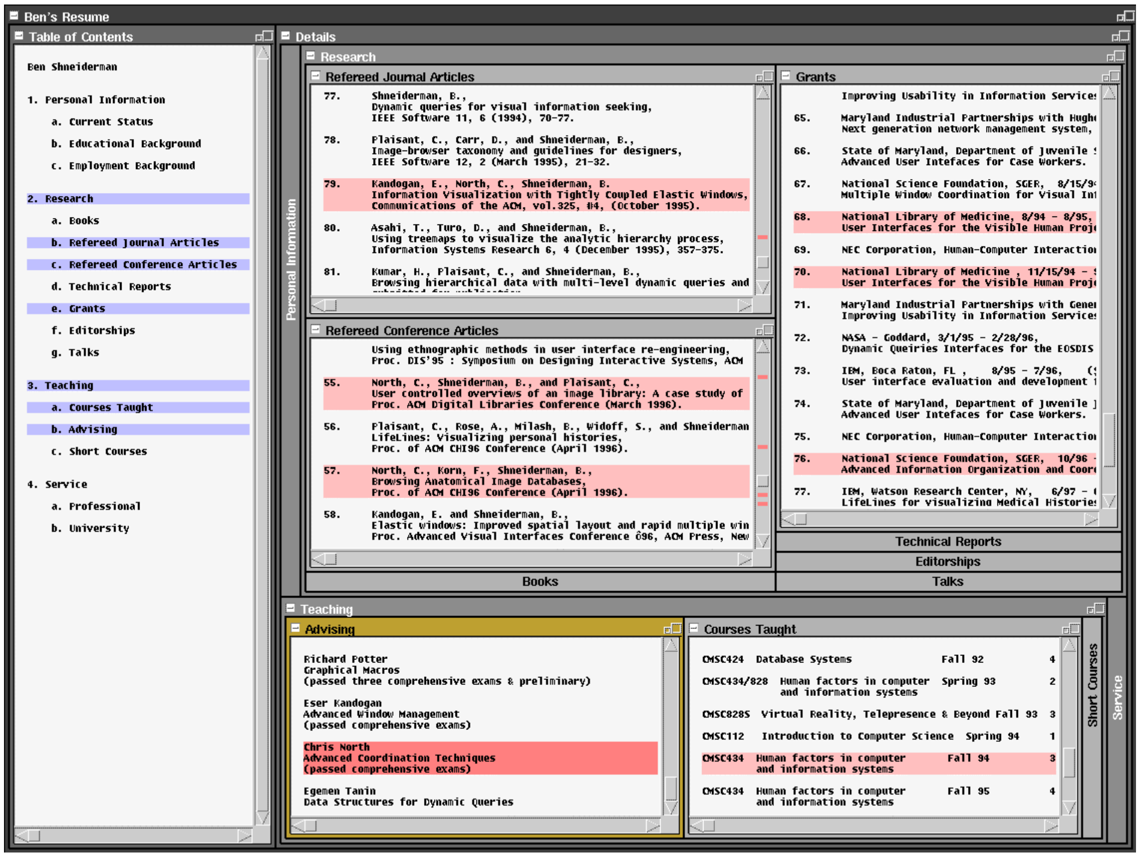


Figure B.4: Attribute Explorer

Visage VQE [DRK97] extends brushing to multiple relations. Visualizations containing joins of relations can be brushed if they share a common relation anywhere in their join paths. An early prototype of LinkKit [Nor98] demonstrates brushing across many-to-many joins for exploring authors, publications, and other references (Figure B.5).



```
Ben Shneiderman

1. Personal Information
     a. Current Status
     b. Educational Background
     c. Employment Background

2. Research
     a. Books
     b. Refereed Journal Articles
     c. Refereed Conference Articles
     d. Technical Reports
     e. Grants
     f. Editorships
     g. Talks

3. Teaching
     a. Courses Taught
     b. Advising
     c. Short Courses

4. Service
     a. Professional
     b. University
```

```
77.     Shneiderman, B.,
        Dynamic queries for visual information seeking,
        IEEE Software 11, 6 (1994), 70-77.

78.     Plaisant, C., Carr, D., and Shneiderman, B.,
        Image-browser taxonomy and guidelines for designers,
        IEEE Software 12, 2 (March 1995), 21-32.

79.     Kandogan, E., North, C., Shneiderman, B.
        Information Visualization with Tightly Coupled Elastic Windows,
        Communications of the ACM, vol.325, #4, (October 1995).

80.     Asahi, T., Turo, D., and Shneiderman, B.,
        Using treemaps to visualize the analytic hierarchy process,
        Information Systems Research 6, 4 (December 1995), 357-375.

81.     Kumar, H., Plaisant, C., and Shneiderman, B.,
        Browsing hierarchical data with multi-level dynamic queries and
```

```
        Using ethnographic methods in user interface re-engineering,
        Proc. DIS'95 : Symposium on Designing Interactive Systems, ACM

55.     North, C., Shneiderman, B., and Plaisant, C.,
        User controlled overviews of an image library: A case study of
        Proc. ACM Digital Libraries Conference (March 1996).

56.     Plaisant, C., Rose, A., Milash, B., Widoff, S., and Shneiderman
        LifeLines: Visualizing personal histories,
        Proc. of ACM CHI96 Conference (April 1996).

57.     North, C., Korn, F., Shneiderman, B.,
        Browsing Anatomical Image Databases,
        Proc. of ACM CHI96 Conference (April 1996).

58.     Kandogan, E. and Shneiderman, B.,
        Elastic windows: Improved spatial layout and rapid multiple win
        Proc. Advanced Visual Interfaces Conference ô96, ACM Press, New
```

```
        Improving Usability in Information Service

65.     Maryland Industrial Partnerships with Hugh
        Next generation network management system,

66.     State of Maryland, Department of Juvenile
        Advanced User Intefaces for Case Workers.

67.     National Science Foundation, SGER,  8/15/9
        Multiple Window Coordination for Visual In

68.     National Library of Medicine, 8/94 - 8/95,
        User Interfaces for the Visible Human Proj

69.     NEC Corporation, Human-Computer Interactio

70.     National Library of Medicine , 11/15/94 -
        User Interfaces for the Visible Human Proj

71.     Maryland Industrial Partnerships with Gene
        Improving Usability in Information Service

72.     NASA - Goddard, 3/1/95 - 2/28/96,
        Dynamic Queiries Interfaces for the EOSDIS

73.     IBM, Boca Raton, FL ,    8/95 - 7/96,
        User interface evaluation and development

74.     State of Maryland, Department of Juvenile
        Advanced User Intefaces for Case Workers.

75.     NEC Corporation, Human-Computer Interactio

76.     National Science Foundation, SGER,  10/96
        Advanced Information Organization and Coor

77.     IBM, Watson Research Center, NY,   6/97 -
        LifeLines for visualizing Medical Historie
```

```
Richard Potter
Graphical Macros
(passed three comprehensive exams & preliminary)

Eser Kandogan
Advanced Window Management
(passed comprehensive exams)

Chris North
Advanced Coordination Techniques
(passed comprehensive exams)

Egemen Tanin
Data Structures for Dynamic Queries
```

```
CMSC424   Database Systems                    Fall 92        4

CMSC434/828  Human factors in computer    Spring 93       2
             and information systems

CMSC828S  Virtual Reality, Telepresence & Beyond Fall 93   3

CMSC112   Introduction to Computer Science  Spring 94     1

CMSC434   Human factors in computer        Fall 94        3
          and information systems

CMSC434   Human factors in computer        Fall 95        4
          and information systems
```

Figure B.5: LinkKit prototype in Elastic Windows

B.2 Navigate ↔ Navigate

This coordination tightly couples navigation in one visualization to simultaneous navigation in another visualization. This maintains synchronization of visualizations while navigating (e.g. scrolling, panning, zooming, slicing, traversing, etc.) through correlated information spaces (e.g. Figure B.6)

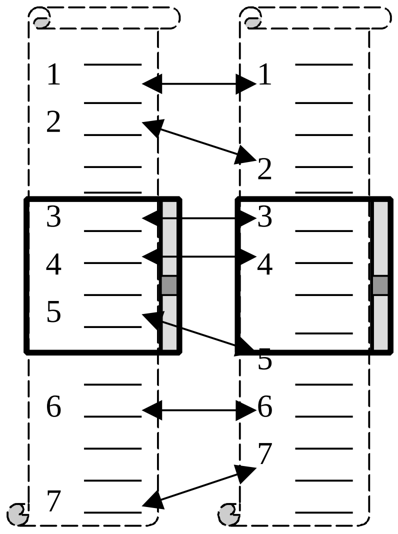
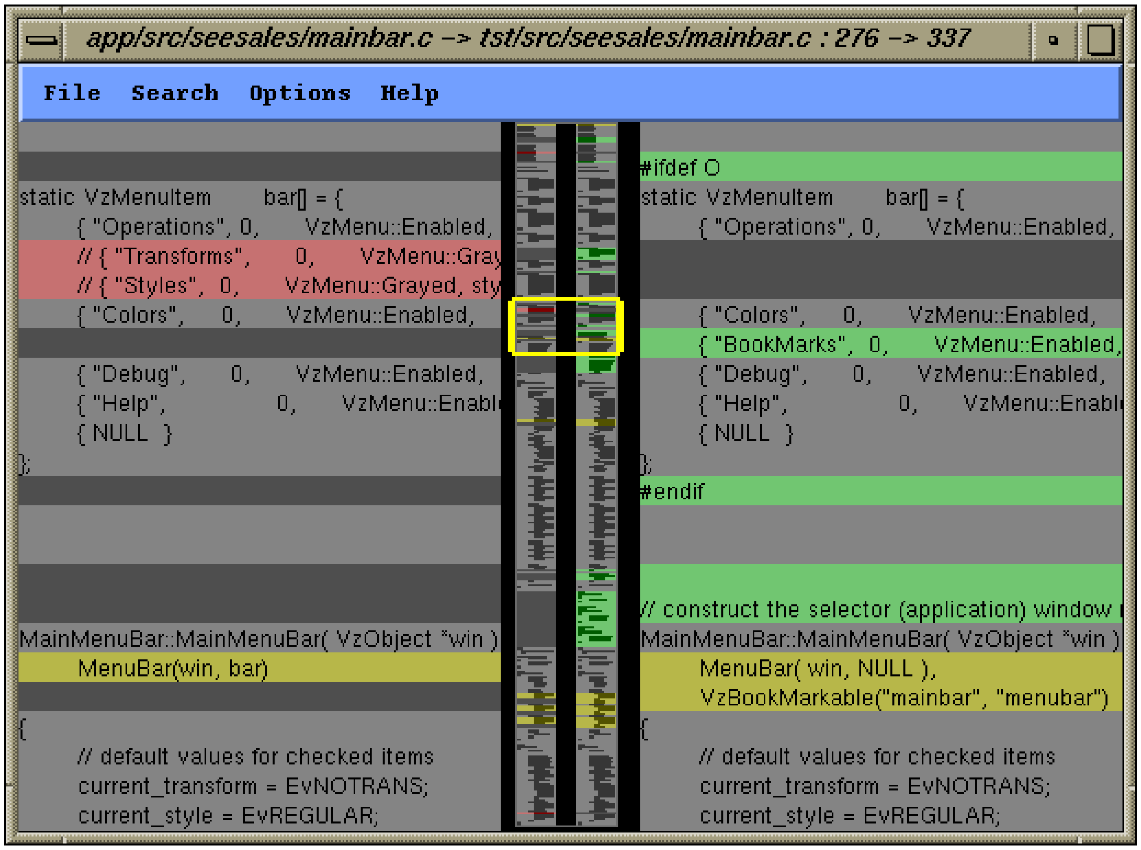


Figure B.6: Synchronized scrolling

Synchronized scrolling tightly couples the scroll bars of two visualizations. WordPerfect displays a document's formatting codes in a separate frame adjacent to the main text that with synchronized scrolling. This approach avoids losing the relationship between representations and saves users from tedious repetition of scrolling actions in each frame. With Logos Bible Software, users can simultaneously scroll through different Bible translations, commentaries, and study guides, which all share a common ordered hierarchical structure of book, chapter, and verse. SeeDiff [BE96] synchronizes scrolling through two version of a source code file for analyzing changes (Figure B.7).

DEVise [LRB97] generalizes this synchronized navigation strategy to 2D, allowing users to synchronously pan and zoom multiple 2D plots with common X and Y axes. The Neighborhood Viewer [CSP97] (Figure B.8) extends this to 3D slicing by synchronously panning correlated cross-section, CT, and MRI images through the human body. Chi et al. [CBR97] (Figure B.9) extends synchronized navigation to general 3D. It arranges many small 3D visualizations in a spreadsheet grid and synchronizes their rotation, zooming, etc.


app/src/seesales/mainbar.c -> tst/src/seesales/mainbar.c : 276 -> 337
File
Search
Options
Help

```
static VzMenuItem       bar[] = {
        { "Operations", 0,      VzMenu::Enabled,
        // { "Transforms",      0,      VzMenu::Gray
        // { "Styles",  0,      VzMenu::Grayed, sty
        { "Colors",     0,      VzMenu::Enabled,

        { "Debug",      0,      VzMenu::Enabled,
        { "Help",               0,      VzMenu::Enabl
        { NULL  }
};



MainMenuBar::MainMenuBar( VzObject *win )
        MenuBar(win, bar)

{
        // default values for checked items
        current_transform = EvNOTRANS;
        current_style = EvREGULAR;
```

```
#ifdef O
static VzMenuItem       bar[] = {
        { "Operations", 0,      VzMenu::Enabled,

        { "Colors",     0,      VzMenu::Enabled,
        { "BookMarks",  0,      VzMenu::Enabled,
        { "Debug",      0,      VzMenu::Enabled,
        { "Help",               0,      VzMenu::Enabl
        { NULL  }
};
#endif


// construct the selector (application) window
MainMenuBar::MainMenuBar( VzObject *win )
        MenuBar( win, NULL ),
        VzBookMarkable("mainbar", "menubar")
{
        // default values for checked items
        current_transform = EvNOTRANS;
        current_style = EvREGULAR;
```

Figure B.7: SeeDiff

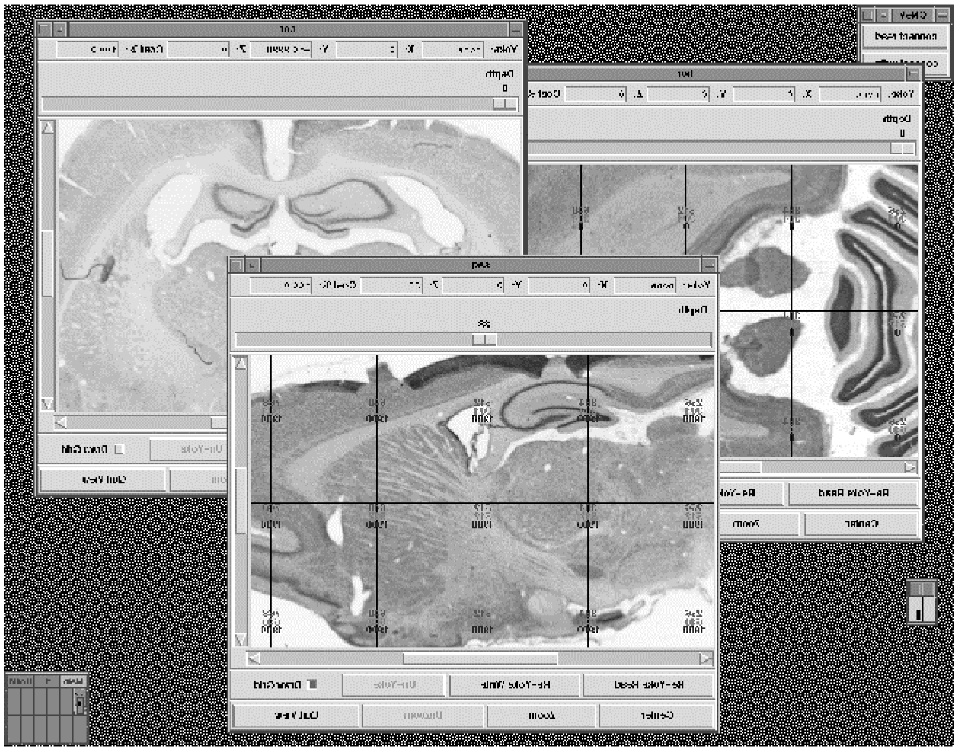
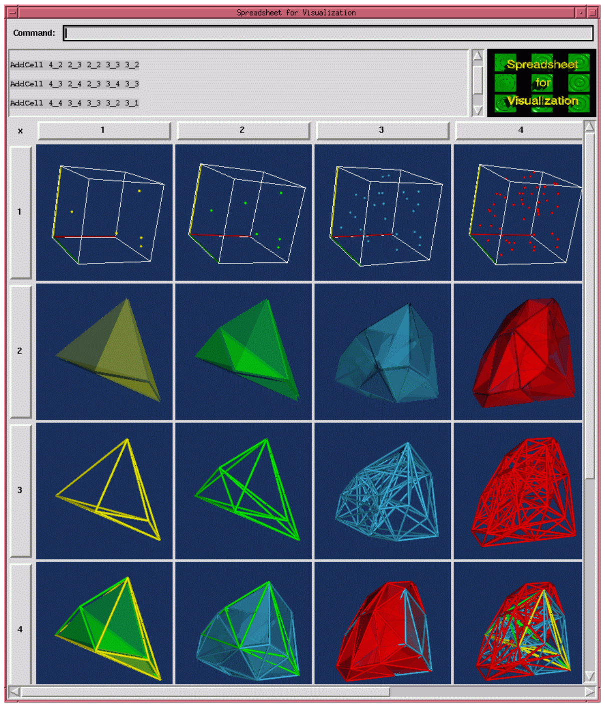


Figure B.8: Neighborhood Viewer



```
AddCell 4_2 2_3 2_2 3_3 3_2
AddCell 4_3 2_4 2_3 3_4 3_3
AddCell 4_4 3_4 3_3 3_2 3_1
```

Figure B.9: Spreadsheet Visualization

B.3 Select ↔ Navigate

This coordination tightly couples selecting items in one visualization to navigating in another visualization, and vice versa (i.e. navigate to select). Users can select items from overviews to navigate to corresponding detailed information in separate visualizations. Likewise, navigating the detailed visualization indicates the corresponding selection in the contextual overview (Figure B.10).

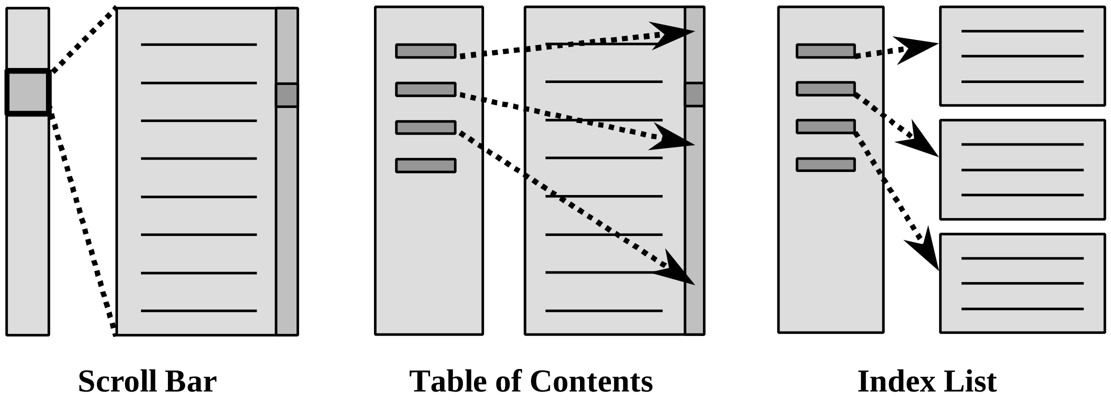


Figure B.10: Overview and detail

Overviews provide a global map of information, and detail visualizations provide detailed information about a small portion. Coordinating the visualizations indicates the location of and provides a mechanism for navigating the detail from within the context of the overview. This is advantageous over detail-only browsers since overviews indicate what information is available, provide context for details, guide browsing, promote exploration, and help avoid getting lost. This strategy contrasts with distortion-oriented techniques [LA94], which attempt to show details within the context of the overview in a single visualization by distorting the view. An important metric is the zoom factor between the overview selection and detail. Larger zoom factors allow for more information. While zoom factors for distortion techniques are typically limited to 5 or less, coordinated visualizations can reach zoom factors of 20 for attribute spaces [PCH92] and 1000 for data aggregation strategies. Also, several of these coordinations can be chained together using intermediate visualizations [PCS95] to multiply zoom factors.

With the Navigational View Builder [MFH95] (Figure B.3), and other web site visualization tools, users can select any node in a visualization of a large site to display that web page in a separate browser window. This strategy has become commonplace in user interface design. It is used in many standard tools such as Microsoft Word and Windows Explorer. It is also used with frames on web pages. Simultaneous menus [HKV00] enables users to select from multiple overviews to display results in a single detail visualization based on all the selections (Figure B.11).

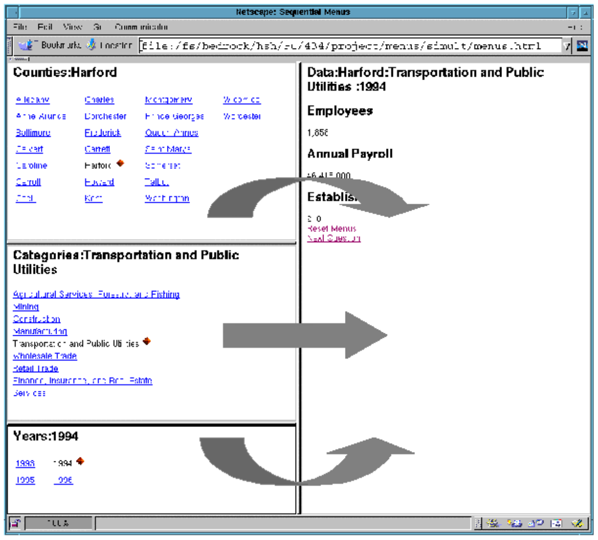
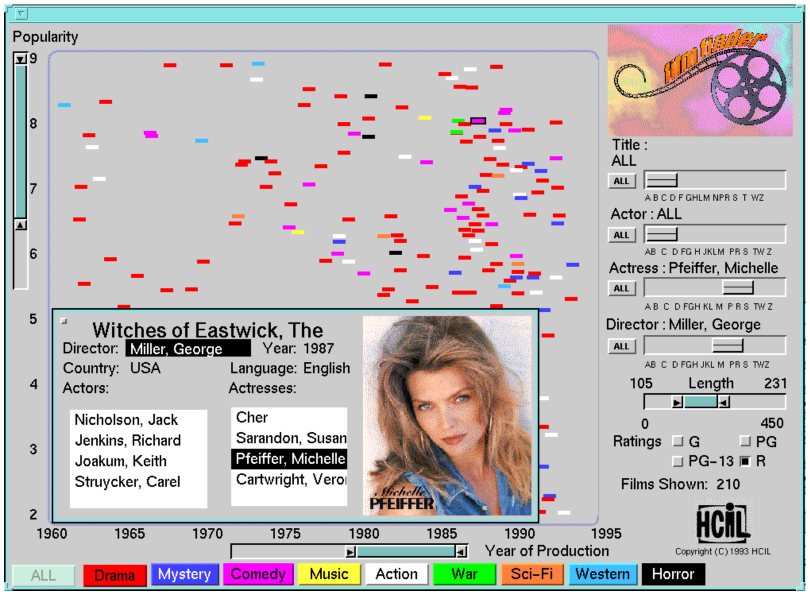

Netscape: Sequential Menus
File Edit View Go Communicator
Bookmarks Location
file:/fs/bedrock/hsh/cc/434/project/menus/simult/menus.html
Counties:Harford
Charles
Dorchester
Montgomery
Worcester
Caroline
Harford
Somerset
Carroll
Howard
Washington
Data:Harford:Transportation and Public Utilities :1994
Employees
1,656
Annual Payroll
Establis
Reset Menus
Categories:Transportation and Public Utilities
Mining
Construction
Manufacturing
Transportation and Public Utilities
Wholesale Trade
Retail Trade
Finance, Insurance, and Real Estate
Services
Years:1994
1993
1994
1995
1996

Figure B.11: Simultaneous Menus

A variant of this approach shows details of selections in a new popup window instead of a given static window, as in the FilmFinder [AS94] (Figure B.12). Selecting a dot on a scatter plot displays that record's fields, including pictures. However, this requires additional clicks to dismiss the popup each time or move it aside.

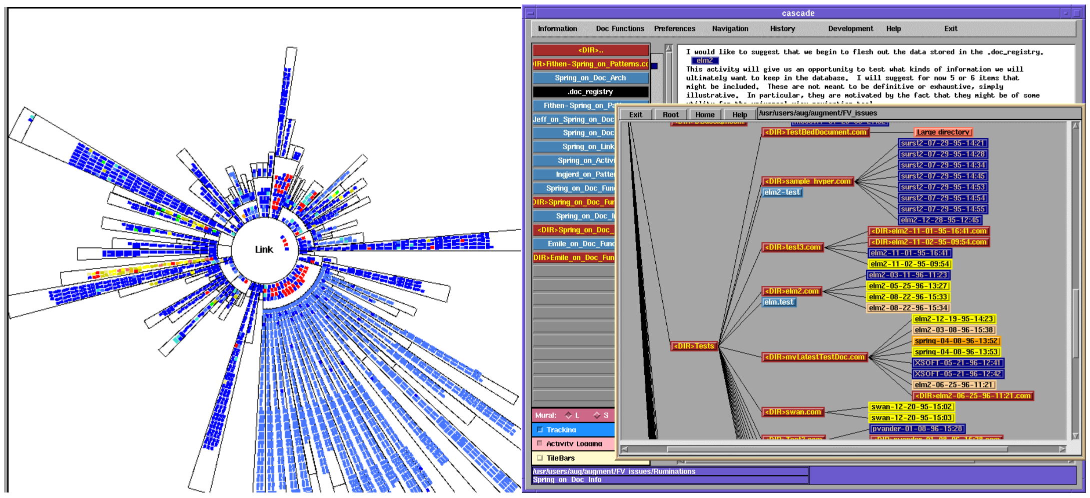


Witches of Eastwick, The
Director: Miller, George Year: 1987
Country: USA Language: English
Actors: Actresses:
Nicholson, Jack
Jenkins, Richard
Joakum, Keith
Struycker, Carel
Cher
Sarandon, Susan
Pfeiffer, Michelle
Cartwright, Veroı

Title : ALL
Actor : ALL
Actress : Pfeiffer, Michelle
Director : Miller, George
105 Length 231
0 450
Ratings G PG PG-13 [✓ R]
Films Shown: 210

Figure B.12: FilmFinder

The select-to-navigate coordination can be used to drill down through layers of a database, with separate visualizations for each layer. CASCADE [SMH96] (Figure B.13) provides four layers of coordinated visualizations for zooming through 4 different levels of scale within a large document database: the Docuverse level (collection of up to 5000 documents), Webview (up to 500 documents), Landmarks (within a single document), and Preview (individual item in a document, such as a hyperlink).

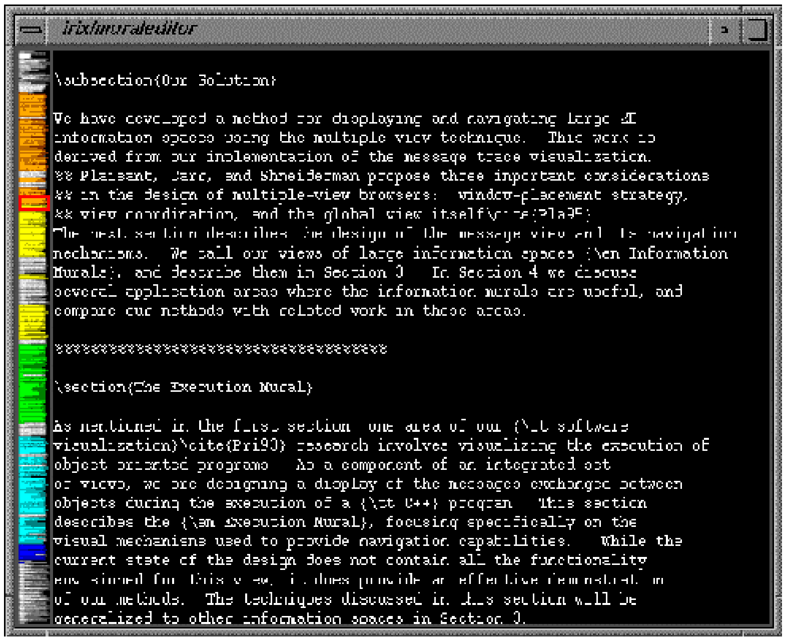


I would like to suggest that we begin to flesh out the data stored in the .doc_registry.
elm2
This activity will give us an opportunity to test what kinds of information we will ultimately want to keep in the database. I will suggest for now 5 or 6 items that might be included. These are not meant to be definitive or exhaustive, simply illustrative. In particular, they are motivated by the fact that they might be of some

Figure B.13: CASCADE

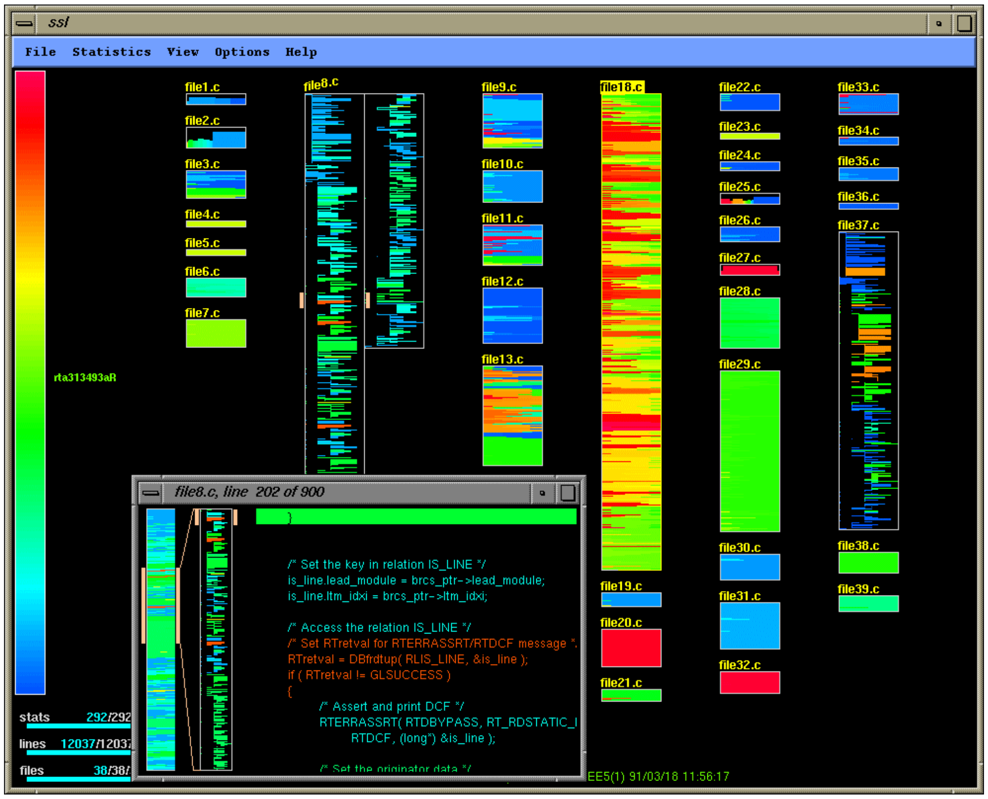
For attribute spaces, dragging or resizing a *field-of-view* indicator (selection) in the overview is tightly coupled to pan or zoom (navigation) the detail visualization, and vice versa. Scroll bars, albeit poor overviews of their associated main window, are a simple 1D example. The Information Mural [JS95] (Figure B.14), SeeSoft [BE96] (Figure B.15), ValueBars [Chi92], and others [Eic94] provide highly reduced images of large documents or software code, using color coding and anti-aliasing algorithms, for navigating 1D document windows with fields-of-view.

The "cursor" link in DEVise [LRB97] links a 2D field-of-view in an overview plot to the panning control of the axes in a detail plot. Similar 2D approaches are used in Pad++ portals [BH94] and in PDQ Trees [KPS97] (Figure B.16) for hierarchies laid out on a 2D surface. Plaisant et al. [PCS95] developed a formal notation for specifying this coordination for browsing large 2D images that is replicated in many digital imaging packages such as Adobe Photoshop.



```
\subsection{Our Solution}

We have developed a method for displaying and navigating large 2D
information spaces using the multiple view technique.  This work is
derived from our implementation of the message trace visualization.
%% Plaisant, Carr, and Shneiderman propose three important considerations
%% in the design of multiple-view browsers:  window-placement strategy,
%% view coordination, and the global view itself\cite{Pla95}.
The next section describes the design of the message view and its navigation
mechanisms.  We call our views of large information spaces {\em Information
Murals}, and describe them in Section 3.  In Section 4 we discuss
several application areas where the information murals are useful, and
compare our methods with related work in these areas.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{The Execution Mural}

As mentioned in the first section, one area of our {\it software
visualization}\cite{Pri93} research involves visualizing the execution of
object oriented programs.  As a component of an integrated set
of views, we are designing a display of the messages exchanged between
objects during the execution of a {\tt C++} program.  This section
describes the {\em Execution Mural}, focusing specifically on the
visual mechanisms used to provide navigation capabilities.   While the
current state of the design does not contain all the functionality
envisioned for this view, it does provide an effective demonstration
of our methods.  The techniques discussed in this section will be
generalized to other information spaces in Section 3.
```

Figure B.14: Information Mural



```
stats      292/292
lines  12037/12037
files       38/38/
```

```
    }

/* Set the key in relation IS_LINE */
is_line.lead_module = brcs_ptr->lead_module;
is_line.ltm_idxi = brcs_ptr->ltm_idxi;

/* Access the relation IS_LINE */
/* Set RTretval for RTERRASSRT/RTDCF message *.
RTretval = DBfrdtup( RLIS_LINE, &is_line );
if ( RTretval != GLSUCCESS )
{
        /* Assert and print DCF */
        RTERRASSRT( RTDBYPASS, RT_RDSTATIC_I
                RTDCF, (long*) &is_line );

        /* Set the originator data */
```

```
EE5(1) 91/03/18 11:56:17
```

Figure B.15: SeeSoft

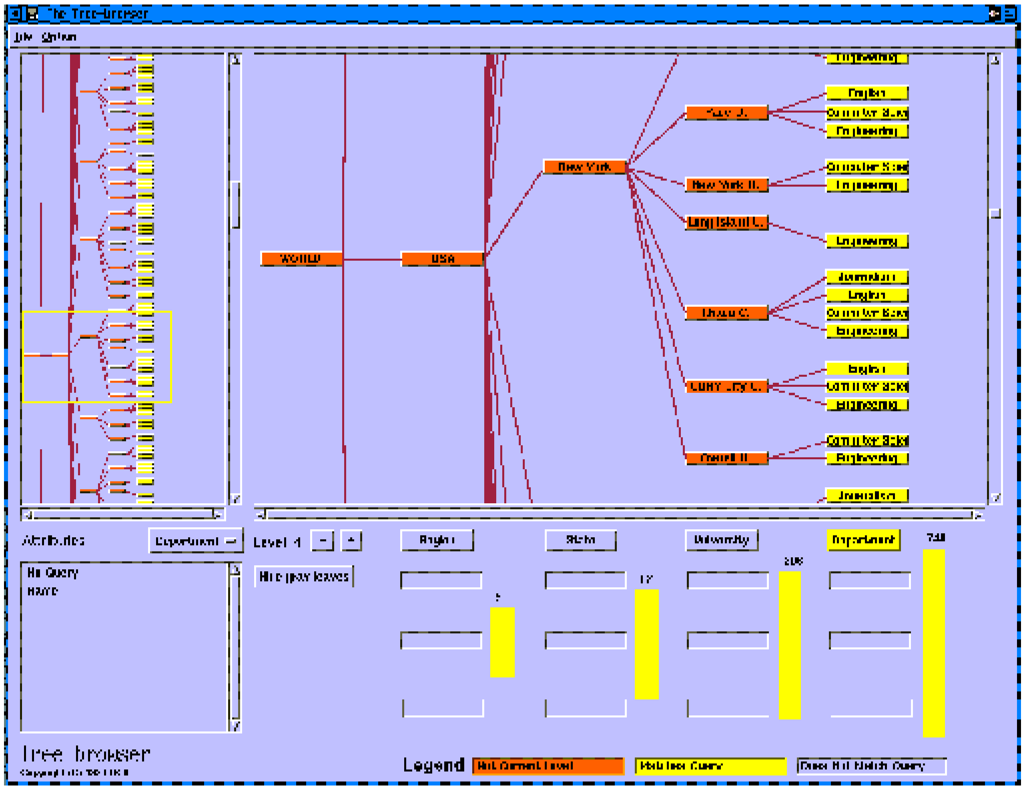


Figure B.16: PDQ Trees

For a 3D volumetric image space, with the Visible Human Explorer [NSP96] users can rapidly navigate each orthogonal 2D cross-section visualization through the human body by dragging the corresponding cut lines in the other visualizations, and receive continuous feedback of contents (Figure B.17).

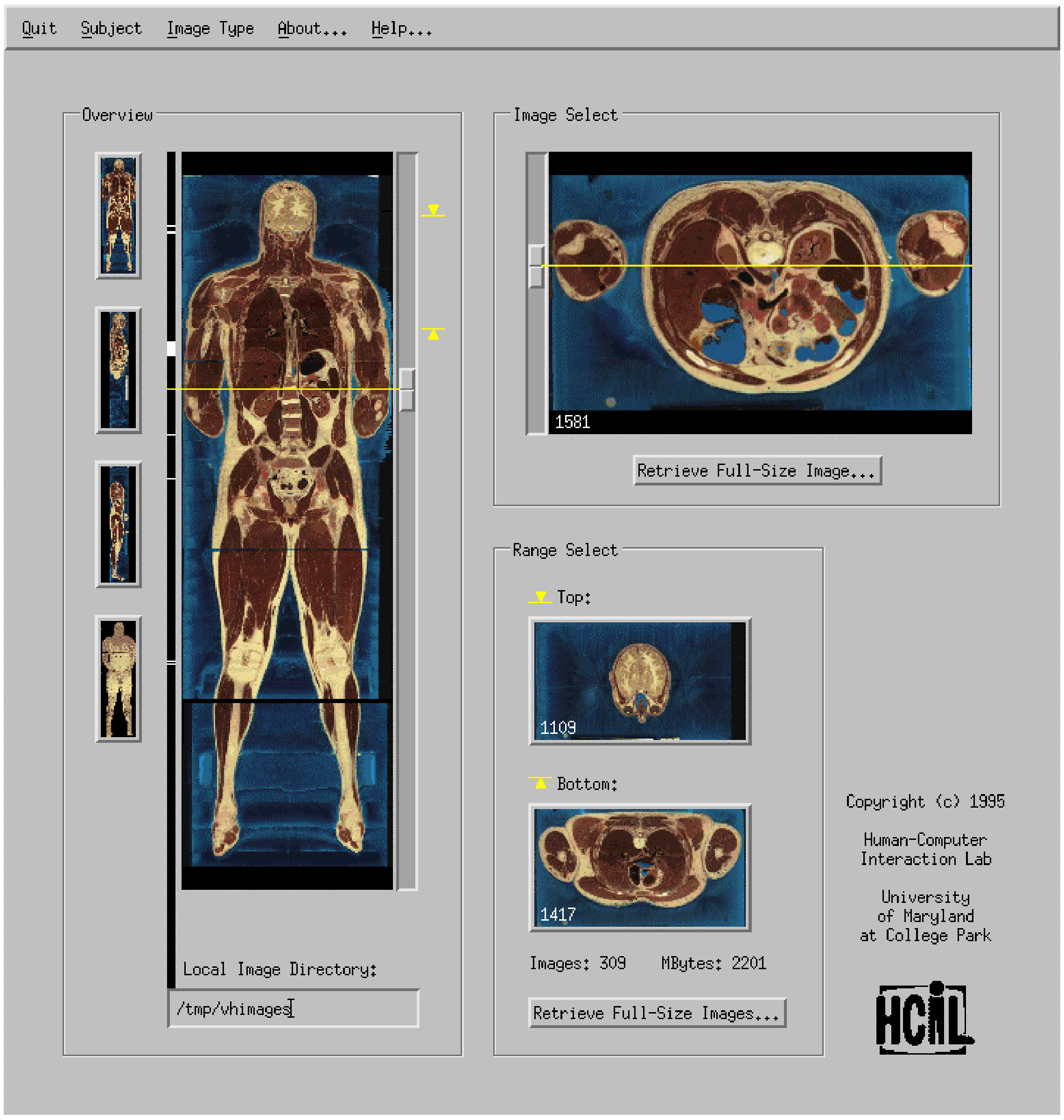


Figure B.17: Visible Human Explorer

An extension to this approach is to use one visualization to keep a history of navigation in other visualizations. With select-to-navigate coordination, users can revisit previous states. PadPrints [HRH98] (Figure B.18) and the Graphical History Browser [AS95] both maintain iconic node-link diagrams of visited web pages for a web browser. Utting and Yankelovich [UY89] review several such approaches for hypertext navigation. They extend their Intermedia system to include a map of destinations that can be reached from the current page as well, hence providing a selectable visualization of both history and potential future.

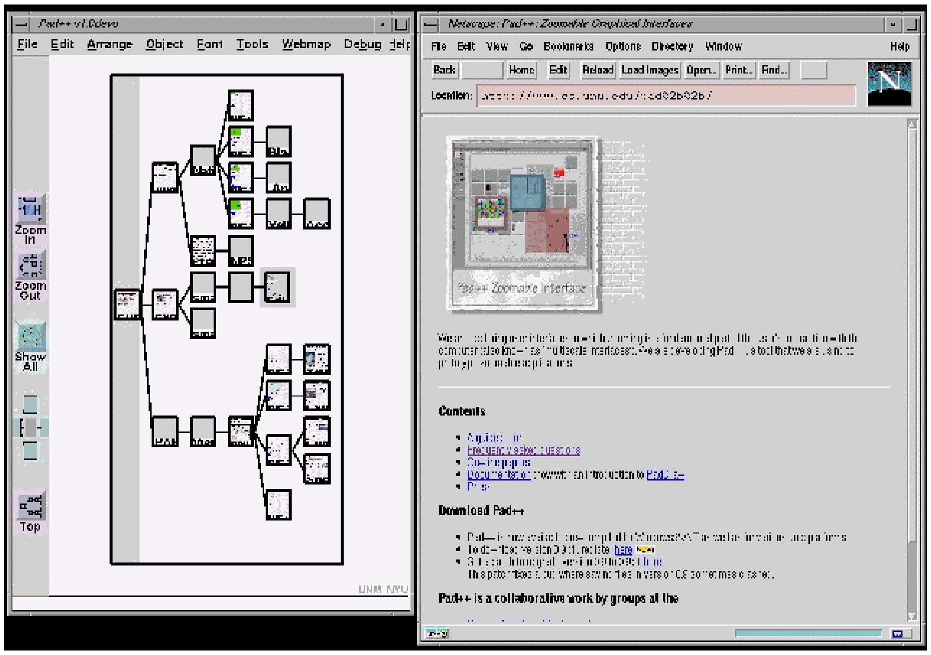

Pad++ v1.0devo
File Edit Arrange Object Font Tools Webmap Debug
Zoom In
Zoom Out
Show All
Top
UNM NYU
Netscape: Pad++: Zoomable Graphical Interfaces
File Edit View Go Bookmarks Options Directory Window Help
Back Home Edit Reload Load Images Open... Print... Find...
Location:
Pad++ Zoomable Interface
Contents
Download Pad++
Pad++ is a collaborative work by groups at the

Figure B.18: PadPrints

B.4 Summary

Many coordinated-visualization interfaces have been developed, and have proven to be very useful and effective. Yet, these are only a small number in comparison to the myriad different combinations of visualizations and coordinations that are needed for so many unique users, data, and tasks. Clearly, these many examples serve to point out the need for Snap-Together Visualization.

Appendix C: User Study Materials

C.1 Evaluation of Coordination Construction

C.1.1 Background Survey

1. Occupation (position title)
2. Census data experience
3. Computer usage experience (frequency, applications)
4. Relational database concepts (tables, attributes, rows, relationships, keys)
5. Microsoft Access experience, SQL experience (designing DBs, writing queries)
6. Visualization tools experience
7. Programming experience (components, databases, user interfaces, web design)

C.1.2 Verbal Post-Survey

1. Other ideas for browsing this data?
2. Trouble spots in using Snap?
3. Suggestions for improving the Snap user interface?

C.2 Evaluation of Coordination Operation

C.2.1 Data

The information presented to the user in the detail window consisted of the following statistics for 47 states. The three missing states were Minnesota, Mississippi, and Missouri.

State:	**Maryland**
Population:	4781468
Families:	1256327
Households:	1749342
Male %:	48.5%
Female %:	51.5%
Urban %:	81.3%

Average Age:	33.1
HS Diploma %:	78.4%
College Degree %:	31.7%
English Speaking %:	84.3%
Average Commute Time:	33
Carpool Commute %:	15.2%
Public Transportation %:	8.1%
Per Capita Income:	17730
Median Family Income:	45034
Median Household Income:	39386
No Income Households %:	15.3%
Average Persons per Family:	3.81
Average Workers per Family:	1.88
Housing Units:	1891917
Vacancy %:	8.2%
Average Bedrooms:	2.73
Average Persons per Unit:	2.73
Median Value:	115500
Median Mortgage:	919
Median Rent:	548
Rent % Household Income:	25.4

Flag Description: The Maryland flag contains the family crest of the Calvert and Crossland families. Maryland was founded as an English colony in 1634 by Cecil Calvert, the second Lord Baltimore. The black and Gold designs belong to the Calvert family. The red and white design belongs to the Crossland family.

C.2.2 Task Sets

Practice Tasks:

What is the Population of Georgia?
Does the information include statistics about the state of Wyoming?
Which of the first four states has the highest xxxx?

Task Group #1:

Question	**Answer**
1. What is the Population of Tennessee?	4,877,185
2. Does the information include statistics about the state of Ohio?	Yes
3. Which of the following states has higher Median Family Income: California or Washington?	CA
4. Which state has Average Commute Time of 31?	NJ
5. How many states in the list begin with the letter 'M'?	5
6. Which of the following 5 states has the highest Median Household Income: Florida, Rhode Island, Louisiana, Alaska, or New Jersey?	Alaska

7. Does the information include statistics about the state of Minnesota?	No
8. Which state has the highest HS Diploma %?	Alaska
9. What is the Population of the 6th state from the bottom of the list?	VT 562,758

Task Group #2:

Question	Answer
1. What is the Population of Texas?	16,986,510
2. Does the information include statistics about the state of Oklahoma?	Yes
3. Which of the following states has higher Median Family Income: Colorado or West Virginia?	CO
4. Which state has Average Commute Time of 32?	NM
5. How many states in the list begin with the word ‘New’?	4
6. Which of the following 5 states has the highest Median Household Income: Georgia, South Carolina, Maine, Arizona, or New Mexico?	GA
7. Does the information include statistics about the state of Mississippi?	No
8. Which state has the highest College Degree %?	Mass
9. What is the Population of the 5th state from the bottom of the list?	VA 6,187,358

Task Group #3:

Question	Answer
1. What is the Population of Utah?	1,722,850
2. Does the information include statistics about the state of Oregon?	Yes
3. Which of the following states has higher Median Family Income: Connecticut or Wisconsin?	Conn
4. Which state has Average Commute Time of 35?	NY
5. How many states in the list begin with the letter ‘O’?	3
6. Which of the following 5 states has the highest Median Household Income: Hawaii, South Dakota, Maryland, Arkansas, or New York?	MD
7. Does the information include statistics about the state of Missouri?	No
8. Which state has the highest English Speaking %?	WV
9. What is the Population of the 4th state from the bottom of the list?	Wash 4,866,692

C.2.3 Questionnaire for User Interface Satisfaction

User Interface #1:

Comprehensibility:	confusing								clear
	1	2	3	4	5	6	7	8	9
Ease of Use:	difficult								easy
	1	2	3	4	5	6	7	8	9
Speed of Use:	slow								fast
	1	2	3	4	5	6	7	8	9
Overall Satisfaction:	terrible								wonderful
	1	2	3	4	5	6	7	8	9

User Interface #2:

Comprehensibility:	confusing								clear
	1	2	3	4	5	6	7	8	9
Ease of Use:	difficult								easy
	1	2	3	4	5	6	7	8	9
Speed of Use:	slow								fast
	1	2	3	4	5	6	7	8	9
Overall Satisfaction:	terrible								wonderful
	1	2	3	4	5	6	7	8	9

User Interface #3:

Comprehensibility:	confusing								clear
	1	2	3	4	5	6	7	8	9
Ease of Use:	difficult								easy
	1	2	3	4	5	6	7	8	9
Speed of Use:	slow								fast
	1	2	3	4	5	6	7	8	9
Overall Satisfaction:	terrible								wonderful
	1	2	3	4	5	6	7	8	9

Comments:

C.2.4 Statistical Results

Effect	F value	p
User Interface	F(2,442) = 86.2	p < .001
Task	F(8,442) = 377	p < .001
UI x Task Interaction	F(16,442) = 20.9	p < .001

Figure C.1: Overall 3x9 ANOVA statistics for user performance

	Task								
	1	2	3	4	5	6	7	8	9
Detail-Only	9.2 *3.0*	8.9 *3.0*	16.6 *5.3*	12.5 *4.7*	10.2 *5.1*	21.2 *5.0*	79.9 *33.2*	86.0 *23.3*	172 *32.5*
No-Coordination	▨	▨	▨	9.9 *2.7*	10.5 *4.5*	22.2 *6.0*	77.9 *34.7*	90.1 *40.0*	174 *41.5*
Coordination	▨	▨	▨	▨	▨	▨	▨	▨	▨
3x1 ANOVA F(2,34), p<	92.9 .001	89.5 .001	86.9 .001	12.5 .001	29.4 .001	23.7 .001	12.2 .001	24.9 .001	47.7 .001

Figure C.2: Mean and *standard deviation* of user performance times (in seconds)

Figure C.2 shows the significance levels of the one-way ANOVAs for the user interface factor for each task. It also shows results of the pair-wise t-test comparisons of user interface treatments within each task. The shaded cells are significantly faster than the white cells within each task at the p<.005 level. The details of these pair-wise comparisons are shown in Figure C.3, the output of the analysis using the SPSS statistical software package. The user-interface treatments and task treatments are coded as follows:

User-Interface Treatments	
1	Detail-Only
2	No-Coordination
3	Coordination

Task Treatments	
1	Coverage-yes
2	Coverage-no
3	Overview patterns
4	Visual lookup
5	Nominal lookup
6	Compare-2
7	Compare-5
8	Search
9	Scan

Task	UI (I)	UI (J)	Mean Difference (I-J)	Std. Error	Sig.[a]	99% Confidence Interval for Difference[a]	
						Lower Bound	Upper Bound
Task 1	1	2	6.894*	.633	.000	5.059	8.730
		3	7.544*	.654	.000	5.649	9.440
	2	1	-6.894*	.633	.000	-8.730	-5.059
		3	.650	.548	.252	-.938	2.238
	3	1	-7.544*	.654	.000	-9.440	-5.649
		2	-.650	.548	.252	-2.238	.938
Task 2	1	2	6.472*	.710	.000	4.413	8.531
		3	6.372*	.570	.000	4.720	8.024
	2	1	-6.472*	.710	.000	-8.531	-4.413
		3	-.100	.304	.746	-.982	.782
	3	1	-6.372*	.570	.000	-8.024	-4.720
		2	.100	.304	.746	-.782	.982
Task 3	1	2	13.344*	1.355	.000	9.419	17.270
		3	13.133*	1.250	.000	9.511	16.755
	2	1	-13.344*	1.355	.000	-17.270	-9.419
		3	-.211	.798	.795	-2.524	2.102
	3	1	-13.133*	1.250	.000	-16.755	-9.511
		2	.211	.798	.795	-2.102	2.524
Task 4	1	2	2.639	1.310	.060	-1.157	6.435
		3	6.783*	1.583	.000	2.197	11.370
	2	1	-2.639	1.310	.060	-6.435	1.157
		3	4.144*	1.187	.003	.703	7.586
	3	1	-6.783*	1.583	.000	-11.370	-2.197
		2	-4.144*	1.187	.003	-7.586	-.703
Task 5	1	2	-.267	1.318	.842	-4.087	3.554
		3	7.628*	1.125	.000	4.367	10.889
	2	1	.267	1.318	.842	-3.554	4.087
		3	7.894*	1.045	.000	4.867	10.922
	3	1	-7.628*	1.125	.000	-10.889	-4.367
		2	-7.894*	1.045	.000	-10.922	-4.867
Task 6	1	2	-.983	1.183	.417	-4.411	2.444
		3	7.250*	1.177	.000	3.838	10.662
	2	1	.983	1.183	.417	-2.444	4.411
		3	8.233*	1.527	.000	3.807	12.660
	3	1	-7.250*	1.177	.000	-10.662	-3.838
		2	-8.233*	1.527	.000	-12.660	-3.807
Task 7	1	2	2.000	11.242	.861	-30.581	34.581
		3	40.778*	8.098	.000	17.307	64.249
	2	1	-2.000	11.242	.861	-34.581	30.581
		3	38.778*	8.270	.000	14.809	62.747
	3	1	-40.778*	8.098	.000	-64.249	-17.307
		2	-38.778*	8.270	.000	-62.747	-14.809
Task 8	1	2	-4.111	8.470	.634	-28.660	20.437
		3	44.167*	4.483	.000	31.173	57.160
	2	1	4.111	8.470	.634	-20.437	28.660
		3	48.278*	8.993	.000	22.213	74.343
	3	1	-44.167*	4.483	.000	-57.160	-31.173
		2	-48.278*	8.993	.000	-74.343	-22.213
Task 9	1	2	-.722	10.914	.948	-32.354	30.909
		3	86.722*	10.422	.000	56.517	116.928
	2	1	.722	10.914	.948	-30.909	32.354
		3	87.444*	9.506	.000	59.894	114.995
	3	1	-86.722*	10.422	.000	-116.928	-56.517
		2	-87.444*	9.506	.000	-114.995	-59.894

Based on estimated marginal means

*. The mean difference is significant at the .01 level.

a. Adjustment for multiple comparisons: Least Significant Diff (equivalent to no adjustments).

Figure C.3: Pair-wise t-test comparisons of user interfaces on user performance

Effect	**F value**	**p**
User Interface	F(2,187) = 70.2	p < .001
Satisfaction Category	F(3,187) = 9.2	p < .001
UI x Category	F(6,187) = 11.1	p < .001

Figure C.4: Overall 3x4 ANOVA statistics for subjective satisfaction

	Satisfaction Category			
	Comprehen-sibility	Ease of Use	Speed of Use	Overall
Detail-Only	6.4	4.1	2.9	3.3
	2.4	*2.4*	*1.5*	*1.4*
No-Coordination	6.6	5.1	4.8	4.8
	2.2	*1.8*	*1.7*	*1.7*
Coordination	8.3	8.1	8.1	7.9
	1.2	*0.9*	*1.1*	*1.1*
3x1 ANOVA	9.4	33.6	83.4	103.5
F(2,34), p<	.001	.001	.001	.001

Figure C.5: Mean and *standard deviation* of subjective satisfaction ratings

Category	UI (I)	UI (J)	Mean Difference (I-J)	Std. Error	Sig.[a]	99% Confidence Interval for Difference[a] Lower Bound	Upper Bound
COMP	1	2	-.111	.511	.830	-1.591	1.369
		3	-1.833*	.459	.001	-3.164	-.503
	2	1	.111	.511	.830	-1.369	1.591
		3	-1.722*	.449	.001	-3.023	-.422
	3	1	1.833*	.459	.001	.503	3.164
		2	1.722*	.449	.001	.422	3.023
EASE	1	2	-1.000	.443	.037	-2.283	.283
		3	-3.944*	.591	.000	-5.658	-2.231
	2	1	1.000	.443	.037	-.283	2.283
		3	-2.944*	.454	.000	-4.259	-1.630
	3	1	3.944*	.591	.000	2.231	5.658
		2	2.944*	.454	.000	1.630	4.259
SPEED	1	2	-1.833*	.406	.000	-3.011	-.656
		3	-5.167*	.398	.000	-6.321	-4.013
	2	1	1.833*	.406	.000	.656	3.011
		3	-3.333*	.412	.000	-4.528	-2.139
	3	1	5.167*	.398	.000	4.013	6.321
		2	3.333*	.412	.000	2.139	4.528
OVERALL	1	2	-1.556*	.283	.000	-2.375	-.736
		3	-4.611*	.354	.000	-5.636	-3.586
	2	1	1.556*	.283	.000	.736	2.375
		3	-3.056*	.338	.000	-4.035	-2.076
	3	1	4.611*	.354	.000	3.586	5.636
		2	3.056*	.338	.000	2.076	4.035

Based on estimated marginal means

*. The mean difference is significant at the .01 level.

a. Adjustment for multiple comparisons: Least Significant Diff (equivalent to no adjustments).

Figure C.6: Pair-wise t-test comparisons of user interfaces on subjective satisfaction

References

[AS94] Ahlberg, C., Shneiderman, B., "Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays", *Proc. ACM CHI'94 Conference - Human Factors in Computing Systems*, pp. 313-317, (1994).

[AW95] Ahlberg, C., Wistrand, E., "IVEE: An Information Visualization and Exploration Environment", *Proc. IEEE Information Visualization '95*, pp. 66-73, (1995).

[AEP96] Antis, J., Eick, S., Pyrce, J., "Visualizing the structure of relational databases", *IEEE Software*, 13(1): 72-79, (January 1996).

[AS95] Ayers, E. and Stasko, J., "Using graphic history in browsing the World Wide Web", *Proc. Fourth International World Wide Web Conference,* (1995).

[BWK00] Baldonado, M., Woodruff, A., Kuchinsky, A., "Guidelines for Using Multiple Views in Information Visualization", Proc. Advanced Visual Interfaces 2000, (2000).

[BE96] Ball, T., Eick, S., "Software visualization in the large", *IEEE Computer*, 29(4):33-43, (April 1996).

[BW90] Beard, D., Walker, J., "Navigational techniques to improve the display of large two-dimensional spaces", Behaviour & Information Technology, 9(6), pp. 451-466, (1990).

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

[BH94] Bederson, B., Hollan, J., "Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics", *Proc. of ACM UIST'94 - User Interface Software and Technology*, pp. 17-26, (1994).

[Bor86] Borning, A., "Constraint-Based Tools for Building User Interfaces", *ACM Transactions on Graphics*, 5(4), pp. 345-374, (October 1986).

[Bro91] Brooks, K., "Lilac: A Two-View Document Editor", *IEEE Computer*, 24(6), pp. 7-19, (June 1991).

[BCS96] Buja, A., Cook, D., Swayne, D., "Interactive High-Dimensional Data Visualization", *Journal of Computational and Graphical Statistics*, 5(1), pp. 78-99, (1996).

[CMS99] Card, S., Mackinlay, J., Shneiderman, B. (editors), *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann, (1999).

[CPF84] Card, S., Pavel, M., Farrell, J., "Window-based computer dialogues", *Proc. INTERACT '84, First IFIP Conference on Human-Computer Interaction*, London, UK, pp. 355-359, (1984).

[CSP97] Carlis, J., Safonov, A., Perrin, D., Konstan, J., "The Neighborhood Viewer: A Paradigm for Exploring Image Databases", *Proc. Companion of ACM CHI'97 Conference - Human Factors in Computing Systems*, pp. 299-300, (1997).

[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).

[Chi92] Chimera, R., " Value Bars: An Information Visualization and Navigation Tool for Multi-Attribute Listings", *Proc. ACM CHI '92*, pp. 293-294, (1992).

[CS94] 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. 1994).

[CHH99] Cox, K., Hibino, S., Hong, L., Mockus, A., Wills, G., "InfoStill: A Task-Oriented Framework for Analyzing Data Through Information Visualization", Bell Labs technical report, http://www.bell-labs.com/org/11359/spr-vis.html, (1999).

[DRK97] Derthick, M., Roth, S., Kolojejchick, J., "Coordinating Declarative Queries with a Direct Manipulation Data Exploration Environment", *Proc. IEEE Information Visualization '97*, pp. 65-72, (1997).

[DC95] Dewan, P., Choudhary, R., "Coupling the User Interfaces of a Multiuser Program", *ACM Transactions on Computer-Human Interaction*, 2(1), pp. 1-39, (March 1995).

[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).

[Eic94] Eick, S., " Data Visualization Sliders," *Proc. ACM UIST '94*, pp. 119-120, (1994).

[EW95] Eick, S., Wills, G., "High Interaction Graphics", *European Journal of Operations Research*, #81, pp. 445-459, (1995).

[FFT74] Fisherkeller, M., Friedman, J., and Tukey, J., "Prim-9: An Interactive Multidimensional Data Display And Analysis System" Slac-Pub-1408, Stanford Linear Accelerator Center, Stanford, California, (1974). Reprinted in Cleveland, W., McGill, R., eds. *Dynamic Graphics for Statistics*, Wadsworth & Brooks, California, (1988).

[FNP99] Fredrikson, A., North, C., Plaisant, C., Shneiderman, B., "Temporal, Geographical and Categorical Aggregations Viewed through Coordinated Displays: a Case Study with Highway Incident Data", *Proc. ACM CIKM '99 Workshop on New Paradigms in Information Visualization and Manipulation*, (1999).

[Fur86] Furnas, G., "Genralized fisheye views", *Proc. ACM CHI '86*, pp. 16-23, (1986).

[Hae88] Haeberli, P., "ConMan: a visual programming language for interactive graphics", *Proc. ACM SigGraph '88*, pp. 103-111, (1988).

[HRH98] Hightower, R., Ring, L., Helfman, J., Bederson, B., Hollan, J., "Graphical Multiscale Web Histories: A Study of PadPrints", *ACM Conference on Hypertext 1998*, (1998).

[Hil92] Hill, R., "The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications", *Proc. ACM CHI'92*, pp. 335-342, (1992).

[HKV00] Hochheiser, H., Kositsyna N., Ville, G., Shneiderman, B., "Performance Benefits of Simultaneous over Sequential Menus as Task Complexity Increases", to appear in IJHCI, (2000).

[HS99] Hochheiser, H., Shneiderman, B., "Understanding patterns of user visits to web sites: interactive starfield visualizations of WWW log data", *Proceedings ASIS '99 Annual Conference*, (1999).

[HM90] Hudson, S., Mohamed, S., "Interactive Specification of Flexible User Interface Displays", *ACM Transactions on Information Systems*, 8(3): 269-288, (July 1990).

[IDC99] International Data Corporation, "Component Architecture for Rapid Delivery of Web-Based Analytic Applications: The AlphaBlox Approach", www.alphablox.com, (1999).

[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).

[KTS00] Kang, H., Tong, J., Shneiderman, B., "Visualization Methods for Personal Photo Collections: Browsing and Searching in the PhotoFinder", University of Maryland, Computer Science Dept. Technical Report, (March 2000).

[Kon97] Konstan, J., personal communication in reference to [CSP97], (1997).

[KPS97] Kumar, H., Plaisant, C., Shneiderman, B., "Browsing Hierarchical Data with Multi-Level Dynamic Queries and Pruning", *IJHCI*, vol. 46, pp. 103-124, (1997).

[LR96] Lamping, J., Rao, R., "The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies", *Journal of Visual Languages and Computing*, 7(1), pp. 33-55, (1996).

[LA94] Leung, Y., Apperley, M., "A review and taxonomy of distortion-oriented presentation techniques", *ACM Transactions on Computer-Human Interactio*n, 1(2):126–160, 1994.

[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).

[Log93] Logos Research Systems, Inc., *Logos Bible Software User Manual*, http://www.logos.com/, (1993).

[MSB90] McDonald, J., Stuetzle, W., Buja, A., "Painting Multiple Views of Complex Objects", *Proc. ECOOP/OOPSLA'90*, pp. 245-257, (1990).

[MHG00] Mockus, A., Hibino, S., Graves, T., "A Web-Based Approach to Interactive Visualization in Context", *Proc. AVI 2000 Conference*, ACM, (May 2000).

[Mon89] Monmonier, M., "Geographic brushing: Enhancing exploratory analysis of the scatterplot matrix", *Geographical Analysis*, 21(1), pp. 81-84, (1989).

[Moo91] Moore, G., Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers, HarperBusiness, (1991).

[MFH95] Mukherjea, S., Foley, J., Hudson, S., "Visualizing Complex Hypermedia Networks through Multiple Hierarchical Views", *Proc. ACM CHI'95*, pp. 331-337, (1995).

[MMM97] Myers, B., McDaniel, R., Miller, R., Ferrency, A., Faulring, A., Borison, E., Kyle, B., Mickish, A., Klimovitski, A., Doane, P., "The Amulet Environment: New Models for Effective User Interface Software Development", *IEEE Transactions on Software Engineering*, 23(6): 347-365, (June 1997).

[NM91] Nardi, B., Miller, J., "Twinkling lights and nested loops: distributed problem solving and spreadsheet development", *Intl. Journal of Man-Machine Studies*, 34(2): 161-184, (1991).

[New78] Newton, C., "Graphics: from alpha to omega in data analysis", *Proc. Symposium on Graphical Representation of Multivariate Data*, Wang, editor, Academic Press, pp. 59-92, (Feb 1978).

[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).

[NSP96] North, C., Shneiderman, B., Plaisant, C., "User Controlled Overviews of an Image Library: A Case Study of the Visible Human", *Proc. ACM Digital Libraries '96 Conference*, ACM Press, pp. 74-82, (1996). Reprinted in *Readings in Information Visualization: Using Vision to Think*, Card, Mackinlay, Shneiderman (editors), Morgan Kaufmann, (1999).

[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).

[Nor98] North, C., "Robust, End-User Programmable, Multiple-Window Coordination", *Proc. ACM CHI'98 Conference*, pg. 60-61, (1998).

[NS99] North, C., Shneiderman, B., "Snap-Together Visualization" (Video), HCIL Video Report 1999, University of Maryland, Computer Science Dept, (1999).

[NS00a] North, C., Shneiderman, B., "Snap-Together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata", *Proc. Advanced Visual Interfaces 2000*, (May 2000).

[NS00b] North, C., Shneiderman, B., "Snap-Together Visualization: Can Users Construct and Operate Coordinated Views?", to appear in *Intl. Journal of Human Computer Studies*, (2000).

[PCH92] Plaisant, C., Carr, D., Hasegawa, H., “When an intermediate view matters: a 2D browser experiment”, University of Maryland Computer Science Dept Technical Report #2980, (October 1992).

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

[PRR99] Plaisant, C., Rose, A., Rubloff, G., Salter, R., Shneiderman, B., “The design of history mechanisms and their use in collaborative educational simulations”, *Proc. of the Computer Support for Collaborative Learning, CSCL' 99*, pp. 348-359, (May 1999).

[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).

[Shn98] Shneiderman, B., Designing the User Interface: Strategies for Effective Human-Computer Interaction, Third Edition, Addison-Wesley, (1998).

[Shn00] Shneiderman, B., “Creating Creativity: User Interfaces for Supporting Innovation”, to appear in *ACM TOCHI, HCI in the Millenium*, ACM, New York, (March 2000).

[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).

[STD95] Spence, R., Tweedie, L., Dawkes, H., Su, H., “Visualisation for Functional Design”, *Proceedings Information Visualization '95*, pp. 4-10, (1995).

[SMH96] Spring, M., Morse, E., Heo, M., “Multi-level Navigation of a Document Space”, *Proc. Leveraging Cyberspace Conference*, (October 1996).

[UY89] Utting, K., Yankelovich, N., “Context and Orientation in Hypermedia Networks”, *ACM Transactions on Information Systems*, 7(1), pp. 58-84, (January 1989).

[Vel88] Velleman, P., *The Datadesk Handbook*, Odesta Corporation, (1988).

[Vin97] Vinoski, S., “CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments”, *IEEE Communications*, 14(2), (Feb. 1997).

[WA95] Ward, M., Allen, M., "High dimensional Brushing for Interactive Exploration of Multivariate Data", *Proc. IEEE Visualization '95*, pp. 271-278, (1995).

[WH87] Wiecha, C., Henrion, M., "Linking Multiple Program Views Using a Visual Cache", *Human-Computer Interaction - INTERACT'87*, pp. 689-694, (1987).

[Wil96] Wills, G., "Selection: 524,288 Ways to Say 'This is Interesting'", *Proc. Information Visualization '96*, IEEE, pp. 54-60, (1996).

[Woo84] Woods, D., "Visual Momentum: a concept to improve the cognitive coupling of person and computer", Int. J. Man-Machine Studies, Vol. 21, 229-244, (1984).