Metavisual Exploration and Analysis of DEVise Coordination in Improvise

Chris Weaver
The GeoVISTA Center and Department of Geography, Penn State University
cweaver@psu.edu

Abstract

Conceptualizing the interactive structure of visualization tools is critical for successful exploration and analysis. As the prominence of multiple coordinated views in visualization increases, better understanding of coordination becomes more and more important for developing improved coordination models that lead to farther reaching applications of coordination. This paper describes how metavisual exploration and analysis of coordination structure and view layout in DEVise visualizations evolved from hand-drawn diagrams to custom-coded software to two Improvise visualizations. The coordination and visual abstraction models implemented in Improvise can be clearly traced back to earlier models implemented in DEVise.

Keywords— **Coordination, multiple views, exploratory visualization, metavisualization.**

1 Introduction

Visualization tools have evolved into integrated, interactive exploration environments in which designers build coordinated multiple view (CMV) visualizations quickly. Unfortunately, interactive complexity grows rapidly as the number of views and amount of coordination increases. Users browsing complex visualizations often have substantial difficulty understanding how CMV visualizations work, in terms of both the layout of multiple views on the screen and the complex web of navigation and selection dependencies between those views.

Tools for visualizing interactive relationships are largely based on graphs. AVS [15] and similar systems use graphs to visualize data flow. The direct manipulation browser in ThingLab/Animus [1] shows a graph representation of the user interface. Interaction Object Graphs (IOGs) allow users to specify widget dependencies using graph-like diagrams [2]. Nodes in IOGs appear as snapshots of widgets in particular interactive states in order to strongly associate on-screen appearance with logical structure. North and Shneiderman [12] have suggested allowing users to build CMV visualizations using a graph representation of coordination structure. Visualization Schemas [11] displays just such a coordination graph in which nodes represent views and edges represent coordinations. In Polaris, nodes in zoom graphs [14] follow a graphical notation that describes a visual query at each vertex of a data cube.

Tools for visualizing screen layout can take advantage of existing visual design approaches such as programming by demonstration [10]. Kandogan and Shneiderman [6] note the importance of layout semantics, especially how partial knowledge of spatial relationships provided by layout can facilitate data exploration. In rendering by example [7], interactive semantics are visualized by overlaying the interface with connecting lines and shared highlighting.

Improvise [16] is a self-contained, fully-interactive environment for building and browsing highly-coordinated visualizations, similar to LinkWinds [5], Tioga-2 [18], Snap-Together Visualization [13], and the Infovis Toolkit [4]. Improvise users create views, access data, specify queries, define visual encodings, and establish coordinations on the fly during exploration and analysis. Moreover, Improvise implements *integrated metavisualization*, using views, lenses, and embedded graphics to reveal the dynamic structure of its own visualizations [17].

DEVise [9] is an interactive editor and browser for visualizing large relational data sets using multiple coordinated scatter plots that have customizable visual encodings. As one of the earliest systems to support construction of CMV visualizations, DEVise is the source of much of the inspiration that went into the development of Improvise. In particular, the coordination and visual abstraction models implemented in Improvise are both descendants of approaches employed in DEVise.

Conversely, Improvise has been used to analyze coordination structure and screen layout in DEVise visualizations. The motivation for visualizing the coordination structure of DEVise visualizations is to provide a clear answer to a simple question: "How does it work?" It is vital to convey coordination semantics to the user as completely and unambiguously as possible, during the visualization design and construction stages as well as the exploration and analysis stages that follow.

The coordination exploration and analysis process has been a self-reinforcing one in which better understanding of coordination applications led to improved coordination models which led to wider-ranging applications of coordi-

nation. This paper recounts this process in the form of a case study that considers:

- the coordination and visual abstraction models implemented in Improvise and DEVise,
- the process of building and browsing Improvise and DEVise visualizations,
- pre-Improvise research efforts to understand coordination in DEVise,
- application of two Improvise visualizations to exploration of DEVise visualizations, and
- how exploration of DEVise coordination structure influenced the design of Improvise.

2 Improvise

Improvise [16] consists of a graphic user interface on top of a modular library of visualization components, forming a fully-implemented, web-capable Java application that appears and behaves like other office productivity applications based on the multiple document desktop metaphor. Users browse visualizations using input gestures to navigate in space and select data items. Although interaction occurs in a single view at any given time, tightly-coupled coordination between views produces the illusion of browsing each visualization interface as a coherent whole. Improvise has been used to build a wide variety of visualizations, each a self-contained XML document file.

Improvise visualizations are built around the Live Properties coordination model and the Coordinated Queries visual abstraction language (figure 1). Live Properties provides a simple, elegant basis for coordinating multiple views through shared interactive parameters that determine how and where views display data. Two *controls* coordinate whenever they are connected through their *properties* via at least one *variable*. Because more than two controls can share the same variable, coordinations are not binary. It is not necessary to connect views pairwise to transitively coordinate sets of views, unlike systems such as DEVise and Snap-Together Visualization [12].

Coordinated Queries is a flexible, yet relatively high-level visualization query language for coordinating data access, processing, and rendering across multiple views. Visual abstraction takes place through expressions that specify how to map data attributes into graphical attributes. Multiple data sets can be loaded, indexed, grouped, filtered, sorted, and projected as a function of interactive navigation and selection in and between multiple views.

The combination of Live Properties and Coordinated Queries in Improvise enables open-ended visual analysis by allowing users to construct and explore highly-coordinated visualizations of multiple simultaneous data

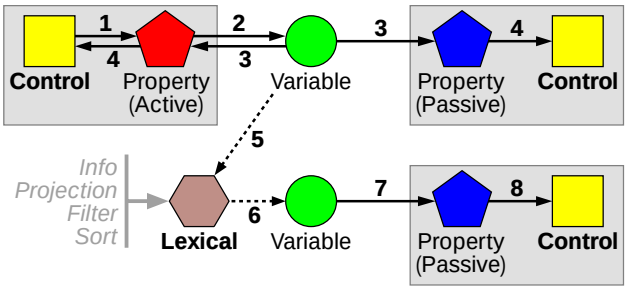

Figure 1: Improvise coordination. In response to interaction, a control modifies the value of one of its properties (1), which assigns the new value to its bound variable (2). The variable sends a notification to all properties bound to it (3), each of which notifies its control of the change (4). The variable also notifies all lexicals whose expressions refer to it (5). Each lexical notifies any variables to which it is assigned as a value (6). When a view receives notification through one of its properties that a lexical has changed (7, 8), it updates itself by processing the modified query.

sets interactively. In the Improvise user interface, designers create, layout, parameterize, and coordinate views. Building occurs in editors inside the same top-level window that contains views. Moreover, building is interactive; changes take effect immediately without the need for a separate compilation stage. This live, amodal interface design allows users to switch rapidly between building and browsing. The goal is to encourage exploration, particularly during initial inspection of newly encountered data sets.

3 DEVise

In DEVise, visualizations consist of scatter plots that act as visual queries on data. Each view shows the rendered records of a single table filtered by the virtual bounds of the view. Figure 2 shows a typical DEVise visualization.

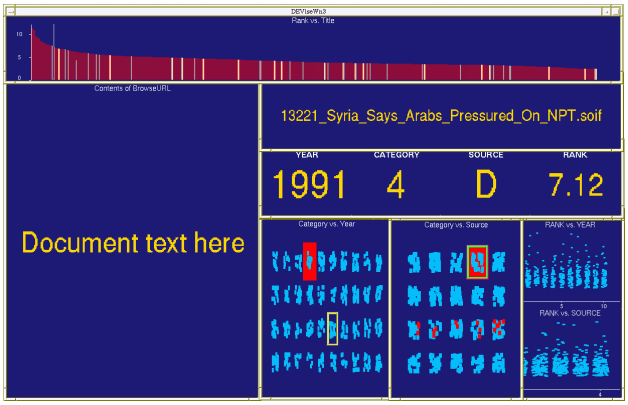

Figure 2: DEVise visualization of ranked news articles.

DEVise has three kinds of coordination: *visual links*, *record links*, and *cursors*. A visual link constrains two views to render identical X and/or Y ranges. A record link renders in a destination view only those records that are visible in a source view. A cursor is a selection box in a view that has the same X and Y ranges as some other view. (Another kind of DEVise link, *piles*, are visual links in which the linked views happen to be stacked on the screen.)

Building visualizations in DEVise is a three-step process. First, the user creates views. Second, the user defines *mappings* that project data attributes into graphical attributes. Each view uses a mapping to draw shapes within it. Third, the user lays out and links views in order to create visual and logical relationships between them. These three steps may be repeated and interleaved as needed. For instance, the user might create a view with a mapping that renders a bar chart of attribute Y vs X using attribute Z as the bar color. The user might then create a second view with a mapping that renders a bar chart of attribute W vs X in the same color scheme. Finally, the user could establish a horizontal-only visual link between the views so that they always show the same range of X values.

Once a DEVise visualization has been built, the user can browse the data interactively by panning and zooming views. However, the user needs to understand how the visualization works in terms of coordination in order to analyze patterns and relationships in the data being displayed. Although DEVise links exhibit an elegant combination of simplicity and flexibility, they are hard to deduce from the appearance of a visualization alone. Learning (and relearning) to interact with a DEVise visualization often requires trial and error, forcing the user to construct a mental model of the linking structure one link at a time. Cyclic chains of links occur frequently, making this task even harder.

4 Early Metavisualization Efforts

4.1 DEVise Layout Manager

The DEVise Layout Manager [8] facilitates design and construction of DEVise visualizations by providing a visual mockup for creation, layout, and linking of views. Screen layout in DEVise is limited to a two-level window hierarchy in which unconstrained top-level windows contain views in an NxM array. In the Layout Manager, views may be unconstrained or connected by *struts* and *rivets*. Views may also contain views in an arbitrary hierarchy.

The Layout Manager uses lines and arrows to represent DEVise linking primitives. These shapes are drawn on top of the windows and views in the visual mockup. For instance, when two views are visually linked on X (that is, they share a range of values on the X axis), a bidirectional yellow arrow indicates the linkage. Figure 3 shows the layout of a DEVise visualization with overlaid link arrows.

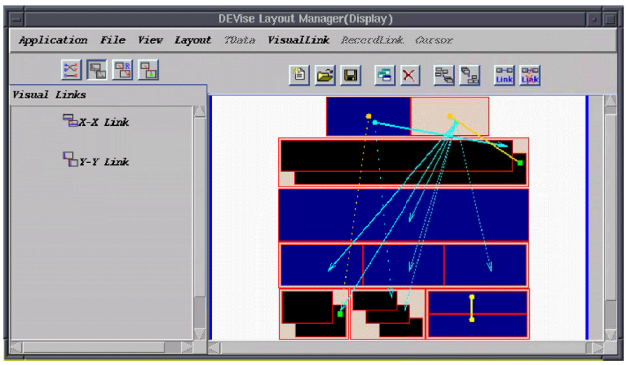

Figure 3: The DEVise Layout Manager, showing the screen layout of the visualization in figure 2. Overlaid arrows indicate synchronized scrolling between views.

As a tool for metavisualizing DEVise visualizations, the Layout Manager has shortcomings. For all but the simplest visualizations, the overlayed graphics obscure each other and the layout beneath. Moreover, because the 2-D location of views constrains how and where overlays can be drawn, overlays cannot depict clearly the multi-dimensional structure typical of DEVise visualizations. These problems are mitigated by the ability to select which types of links appear as overlaid arrows at any given time.

Despite these problems, working with the Layout Manager produced a critical insight: display of coordination can be separated from display of layout. Whereas visualizations of *physical layout* display on-screen spatial relationships between views, visualizations of *logical layout* display coordination structure independently of any on-screen spatial relationships between views.

4.2 Logical Layout Editor

The Logical Layout Editor was created for experimenting with logical layout techniques with the goal of implementing a flexible tool for visual analysis of coordination structure in DEVise visualizations. As such, the visual representation of DEVise coordination graphs changed over multiple successive versions of the Logical Layout Editor.

The first version uses nodes for views and edges for visual links, record links, and cursors. Nodes are drawn as circles and edges are drawn as arrows. In this representation, each set of visually-linked views appears as a fully-connected subgraph of nodes. Similar graph structures emerge from the semantics of record links and cursors: two-layer trees for record linked views, directed pairs for views connected by cursors. The second version adds *packs* as a way to visually cluster nodes in such graph structures. Packs are drawn as balloons—enlarged, filled convex hulls—around the nodes they contain. Around groups of nodes that represent visually linked views, packs

reveal group containment and overlap better than closed sets of pairwise edges, especially for groups with more than four views in which edges are forced to cross.

Seeing logical layouts drawn using packs instead of arrows motivated the third version of the editor (figure 4), in which nodes represent all DEVise primitives (views, visual links, record links, cursors) and edges indicate dependencies between them. The graph substructures that result are strikingly different from those in earlier versions. A set of visually linked views appears as a set of view nodes radiating from a central visual link node. A set of record linked views has a similar appearance, but with one view node connected toward the record link node and the remaining view nodes connected away. A cursor appears as a

$$View1 \rightarrow Cursor \rightarrow View2$$

triple. The graph also reveals the presence of a fourth frequent structure, a *record cursor*, of form

$$View1 \rightarrow Cursor \rightarrow View2 \rightarrow RecordLink \rightarrow View3$$

In this structure, $View2$ is an DEVise implementation artifact that must be hidden offscreen to avoid distraction.

Logical layout depicts coordination structure in even the most complicated DEVise visualizations. For example, the visualization in figure 2 is challenging to comprehend even with assistance from an experienced user. Its structure contains cyclic dependencies which are hard to deduce from seeing physical layout but are obvious in a logical layout graph. The graph readily reveals all four kinds of coordination substructure. Edge direction depicts how manipulation of the spatial extent of views or cursors leads to changes in the spatial extent and data contents of other views.

Logical layout is a useful way to show how to manipulate a DEVise visualization by showing how its views collectively respond to interaction. However, the various versions of logical layout are hard-coded special cases of graph visualization. The first step in building a more flexible tool for exploratory metavisualization of DEVise visualizations is to recreate the logical layout graph using the general-purpose graph view available in Improvise. The second step is to coordinate this view with others so as to reveal additional relationships between DEVise primitives.

5 Metavisualizing DEVise in Improvise

5.1 TGraph

The *tgraph* Improvise visualization, shown in figure 5, statically metavisualizes DEVise visualizations using a graph and a table. The graph displays the coordination structure (logical layout) of the visualization using nodes to represent DEVise views and links, and packs to surround views that are related to each other through common

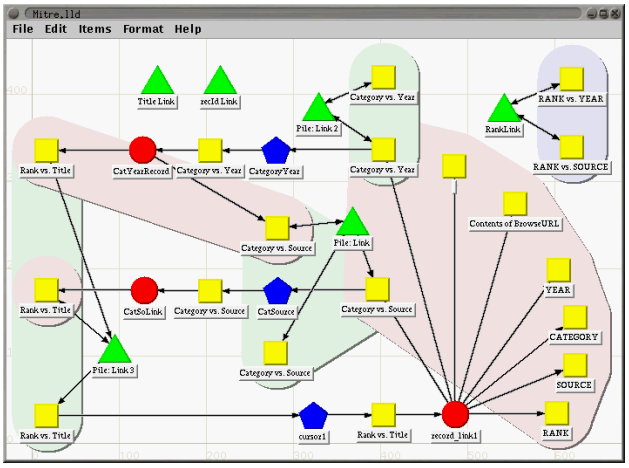

Figure 4: Logical layout graph of the visualization in figure 2. Squares represent views. Triangles, circles, and pentagons represent visual links, record links, and cursors, respectively. Groups of linked nodes are contained in packs drawn in corresponding colors.

links. To provide perceptual coherence, views, visual links, record links, and cursors are uniformly encoded throughout the visualization using different colors.

The graph and table views share a filter query that determines which types of DEVise primitives are shown. The filter is defined using four boolean variables, which the user can toggle on and off using checkbox controls. In the graph view, the filter is bound to the `node filter` property. The `link filter` property is bound to a second filter, defined in terms of five boolean variables that determine which types of dependencies between DEVise primitives are visible as edges. In this way, the user can rapidly ascertain the presence of and relationships between different coordination substructures. In figure 6, for example, the graph has been filtered to show three record cursors by toggling the appropriate checkboxes.

The *tgraph* visualization accesses data using a "session dump" feature in DEVise that writes a node-and-edge graph representation of the DEVise visualization's structure to a file in a tabular format (table 1).

type	*name*
`string`	`ID`
`string`	`Name`
`string`	`Type`
`int`	`X`
`int`	`Y`

Table 1: Schema of records that describe DEVise objects and relationships between them.

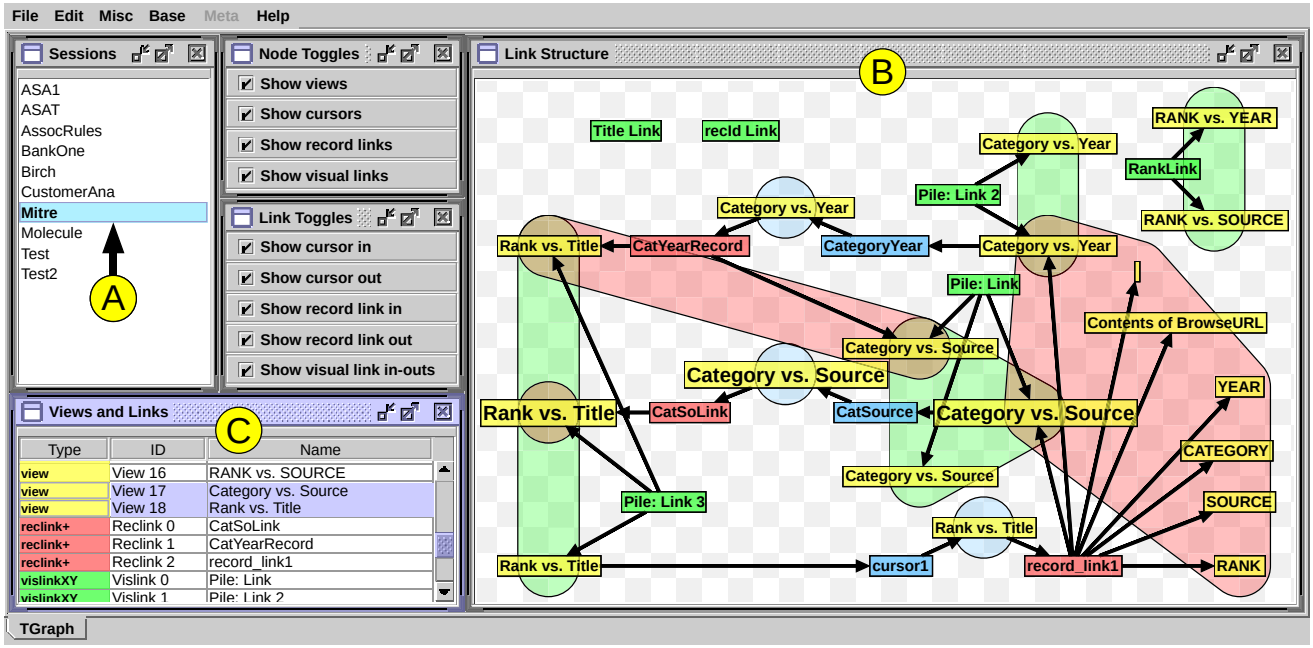

Figure 5: The *tgraph* Improvise visualization. In a list of available visualizations, selecting the DEVise visualization from figure 2 (A) loads its coordination structure into a graph of relationships (B) and a list of objects (C).

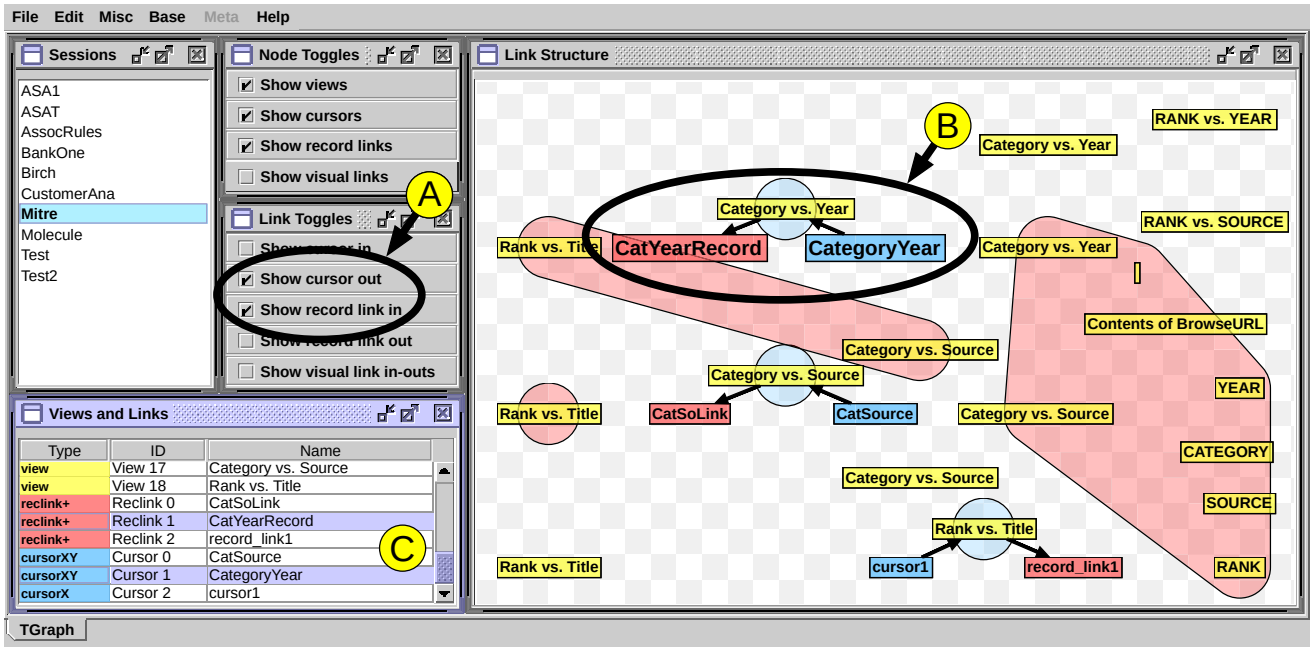

Figure 6: Category filtering in the *tgraph* visualization. Checkboxes (A) toggle boolean variables used to filter objects and relationships in the graph (B) and list (C) views. In this case, filtering reveals the presence of three *record cursor* substructures.

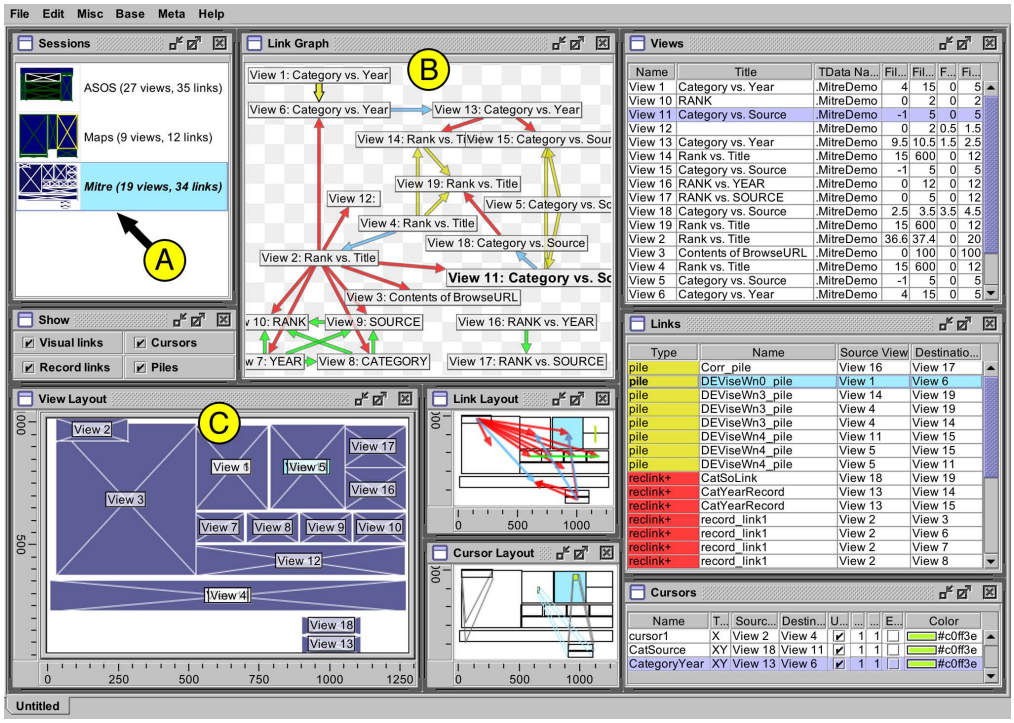

Figure 7: The *devise-mv* visualization. In a list of available DEVise visualizations (shown in miniature), selecting the visualization from figure 2 (A) loads its coordination structure into a graph (B) and its screen layout into a scatter plot (C).

Dumped files contain records for dependencies between DEVise primitives as well as records for the primitives themselves. Pairwise dependencies between a view and another primitive—a visual link, record link, or cursor—are encoded into the fields of records as

$$\{LinkID, srcPrimitiveID, dstPrimitiveID, 0, 0\}$$

For example, `View17` is the source view of `Reclink0` in the record `{Link0, View17, Reclink0, 0, 0}`. Dependencies between groups of views—such destination views in record links and transitively closed sets of views in visual links—are encoded as

$$\{PackID, PackName, ViewID, 0, 0\}$$

For records that represent the primitives themselves, the `X` and `Y` fields are used to determine the initial location of each primitive's node in the graph view.

A master data set that lists available DEVise session dumps is generated from a directory in the local file system. Selecting an item from the master list loads a particular session dump file from that directory. Grouping queries classify records from the file into node, edge, and pack categories used by the view to build graph topology.

The *tgraph* visualization uses filter and visual encoding queries to determine the structure, visibility, and appearance of nodes and edges in the graph view. Graph structure is determined by naming edge endpoints and pack members using unique node identifiers. Visibility of graph elements depends on the state of relevant checkboxes via boolean variables. Node and pack glyphs are colored according to the type of DEVise primitive they represent. Edge glyphs are drawn as black arrows, and do not depend on data attributes to determine their appearance.

The graph view depends on the session list and checkboxes via paths through the *tgraph* visualization's coordinated query graph. Selecting a DEVise visualization in the list causes the graph view to build a new graph from the corresponding node, edge, and pack data sets.

Addition of the node and edge visibility toggles, including layout of the checkboxes and editing of the filters, took only a few minutes. This ability to interactively modify a metavisualization to explore the structure of visualizations using Coordinated Queries is a substantial improvement over logical layouts and other metavisualizations custom-programmed for a particular visualization system.

5.2 DEVise-MV

The *devise-mv* Improvise visualization, shown in figure 7, statically metavisualizes DEVise visualizations using a graph, scatter plot, and several tables. The graph displays the coordination structure of the visualization using

nodes to represent DEVise views and edges to represent links. The scatter plot draws a miniature version of DEVise screen layout using position information in the records that describe DEVise views. To provide perceptual coherence, DEVise primitives are uniformly colored throughout the visualization (as in the *tgraph* visualization).

The graph view and the `Links` table view are filtered using (different) filter queries defined in terms of five boolean variables. The user can toggle these variables using checkboxes (figure 8). The graph view and the `Views` table view are further filtered to show only the DEVise views that are at least partially visible in the main scatter plot. In this way, the user can identify which links affect which parts of the screen in the visualization.

A portal in the main scatter plot is navigationally coupled with link and cursor scatter plots, resulting in a three-way coordination that combines aspects of overview+detail, synchronized scrolling, and small multiples. In figure 9, the portal has been panned to display the right half of the visualization screen layout. Visual links and piles have been made invisible. The resulting metavisualization reveals details about extraneous views that were created to support two record cursors, including their artifactual presence at the bottom of the screen.

The *devise-mv* visualization accesses data generated using a second version of the DEVise "session dump" feature that writes a substantially more detailed description of views, visual links, piles, record links, and cursors in a session. Although the dumped file is written in a record-oriented format, each kind of DEVise primitive has a different description, necessitating variant records. Records of three different schemas, shown in tables 2– 4, are written to the file in the session dump. The records of each schema are separated into two XML-formatted files.

A master data set that lists available DEVise session dumps is generated from a directory in the local file system. Selecting an item from the master list loads its pair of XML-formatted files from that directory. The graph view uses the resulting data sets to construct the node-and-edge topology of its graph. The same data sets are separately displayed in the view and link tables.

The *devise-mv* visualization uses filter and visual encoding queries to determine the structure, visibility, and appearance of nodes and edges in the graph view. Graph structure is determined by naming edge endpoints using unique node identifiers. The direction of each edge is determined by the type of DEVise link it represents. Node visibility depends on whether or not the views they represent are contained within the portal in the main scatterplot. Edge visibility depends on the state of relevant checkboxes via boolean variables. Each node glyph displays the name and type of the view it represents. Each edge glyph is drawn as an arrow colored according to the type of the DEVise link it represents.

type	*name*
string	Name
string	Title
string	Parent View
color	Foreground Color
color	Background Color
string	Pile
int	Dimensions
string	TData Name
int	X
int	Y
int	Width
int	Height
double	Filter X Lo
double	Filter X Hi
double	Filter Y Lo
double	Filter Y Hi

Table 2: Schema of records that describe DEVise views.

type	*name*
string	Name
string	Type
string	Source View
string	Destination View
boolean	Use Grid
int	Grid X
int	Grid Y
boolean	Edge Grid
color	Color

Table 3: Schema of records that describe DEVise cursors.

type	*name*
string	Name
string	Type
string	Source View
string	Destination View

Table 4: Schema of records that describe DEVise record links, visual links, and piles.

The graph view depends on the session list, main scatterplot, and checkboxes via paths through the *devise-mv* visualization's coordinated query graph. Selecting a DEVise visualization in the list causes the graph view to rebuild the graph from that session's data set; it also causes the main scatter plot and portal to display the newly selected data set. Dragging and stretching the portal horizontally

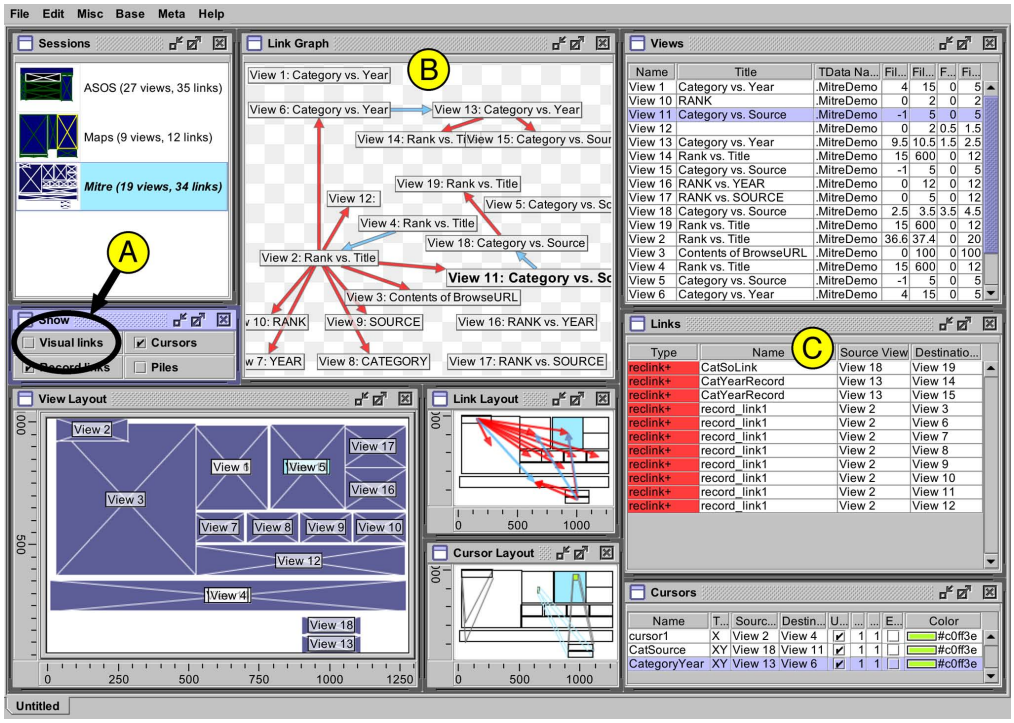

Figure 8: Category filtering in the *devise-mv* visualization. Checkboxes (A) toggle boolean variables that affect the visibility of different kinds of DEVise links that are represented as edges in the graph (B) and rows in the links table (C).

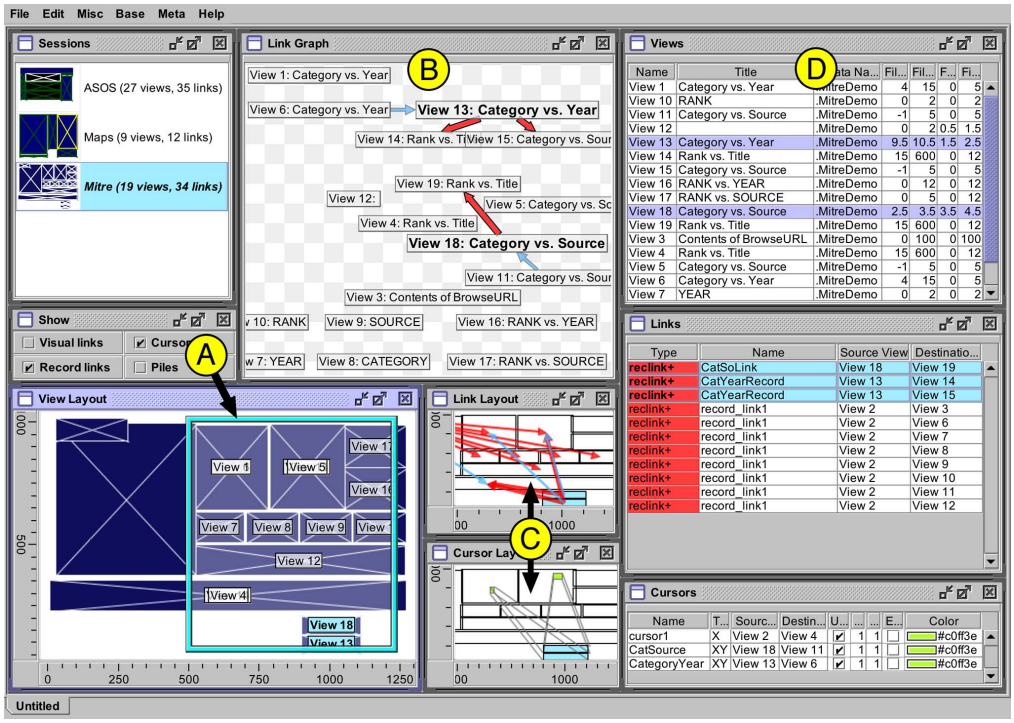

Figure 9: Spatial filtering in the *devise-mv* visualization. Moving a portal above the screen layout (A) causes the graph (B) and views table (D) to show only views inside the portal bounds. The portal is navigationally coordinated with two scatter plots that display the screen layout of views overlaid with link and cursor relationships (C).

affects filtering of nodes in the graph view. Toggling the boolean value of the `Visual Link Flag` variable (using the corresponding checkbox) affects filtering of edges in the graph view.

In lieu of explicit relational join queries, Coordinated Queries provides cross-table indexing, thereby enabling the kind of complex filters used in this metavisualization. For instance, the `Links` and `Cursors` table views show only the DEVise primitives that link views which are contained in the portal in the main layout scatter plot. To do this, it is necessary to filter on screen position information (in the views data set) indexed using source and destination view names (in the links and cursors data sets). A similar technique is used to draw the topmost layer in the `Links` scatter plot, by drawing links and cursors as colored arrows between the centers of the rectangles that represent views, thereby recreating the appearance of the Layout Manager.

The process of building the *tgraph* and *devise-mv* visualizations reveals a limitation in the current design of Improvise graphs: the inherent structure of accessed data sets (like the session dumps) limits the ways in which those data sets may be projected into nodes and edges in graphs, and how those nodes and edges may be filtered. This is why the *tgraph* visualization shows all DEVise primitives as nodes and dependencies between them as edges, while the *devise-mv* metavisualization shows DEVise views as nodes and other primitives as edges.

6 Evolution of DEVise Linking Structure

The earliest attempts at metavisualization involved both hand-drawn and custom-coded views of DEVise coordination structure. Despite the crudeness of these approaches, they led to the discovery that DEVise views and links can be translated into a simpler, more flexible set of coordination and visual abstraction components. These components are the basis of Live Properties and Coordinated Queries. Figure 10 shows translation stages for visual links, cursors, record links, and view mappings.

Translation of visual links into Coordinated Queries can be broken down into four stages:

1. In DEVise, visual links (L_{12}, L_{13}, L_{23}) form a transitive closure of symmetric connections between scatter plots (V_1, V_2, V_3).

2. The transitive closure can be treated as a single coordination (L) connecting multiple views.

3. The coordination is a special case of synchronized scrolling in which views always display the same rectangular region of the cartesian plane. This rectangular region can be defined as a pair of ranges on X and Y ($R_X \times R_Y$).

4. In Improvise, multiple views bind to the same pair of independent range variables (R_X, R_Y).

Translation of visual links reveals that synchonized scrolling can be treated as two distinct coordinations involving separable navigation in orthogonal (horizontal and vertical) dimensions. This separation of scrolling dimensions increases coordination flexibility by allowing views to synchronize horizontally, vertically, or in a crossed fashion, as well as in the usual two-dimensional manner.

Translation of cursors into Coordinated Queries can be broken down into five stages:

1. In DEVise, a cursor (L) connects the full extent of one view (V_2) to an arbitrary rectangular subregion of another view (V_1).

2. The connection involves two relationships: (1) containment of a movable portal (P) inside V_1, and (2) the equivalent of a visual link between P and V_2.

3. The combination of relationships is a special case of overview+detail in which the detail view (V_2) coordinates with a control nested inside the spatial context of the overview (V_1) rather than the overview itself.

4. The rectangular overview and detail regions can be defined as two pairs of ranges on X and Y ($R_{CX} \times R_{CY}$ and $R_X \times R_Y$, respectively).

5. In Improvise, a rectangular portal (P) control is nested in a scatter plot (V_1). Two contextual range properties bind to the overview's bound range variables (R_{CX}, R_{CY}). Two additional range properties bind to the range variables (R_X, R_Y) shared with the detail view (V_2).

Translation of cursors reveals that overview+detail can be treated as two distinct navigation coordinations: (1) between a parent view and a nested child control, and (2) between the child control and other views. Moreover, doing so increases coordination flexibility by enabling several useful variations on overview+detail, including 1-D and autocorrelation ($X = Y$) sliders.

Translation of record links into Coordinated Queries can be broken down into four stages:

1. In DEVise, record links (L) filter (f) the contents of one view (V_2) to show only those records that are at least partially visible in another view (V_1).

2. The rectangular extent of V_1 can be defined as a pair of ranges on X and Y ($R_X \times R_Y$).

3. The filter can be expressed as a function of two range variables (R_X, R_Y). The filter passes only those records for which $x \in R_X$ and $y \in R_Y$.

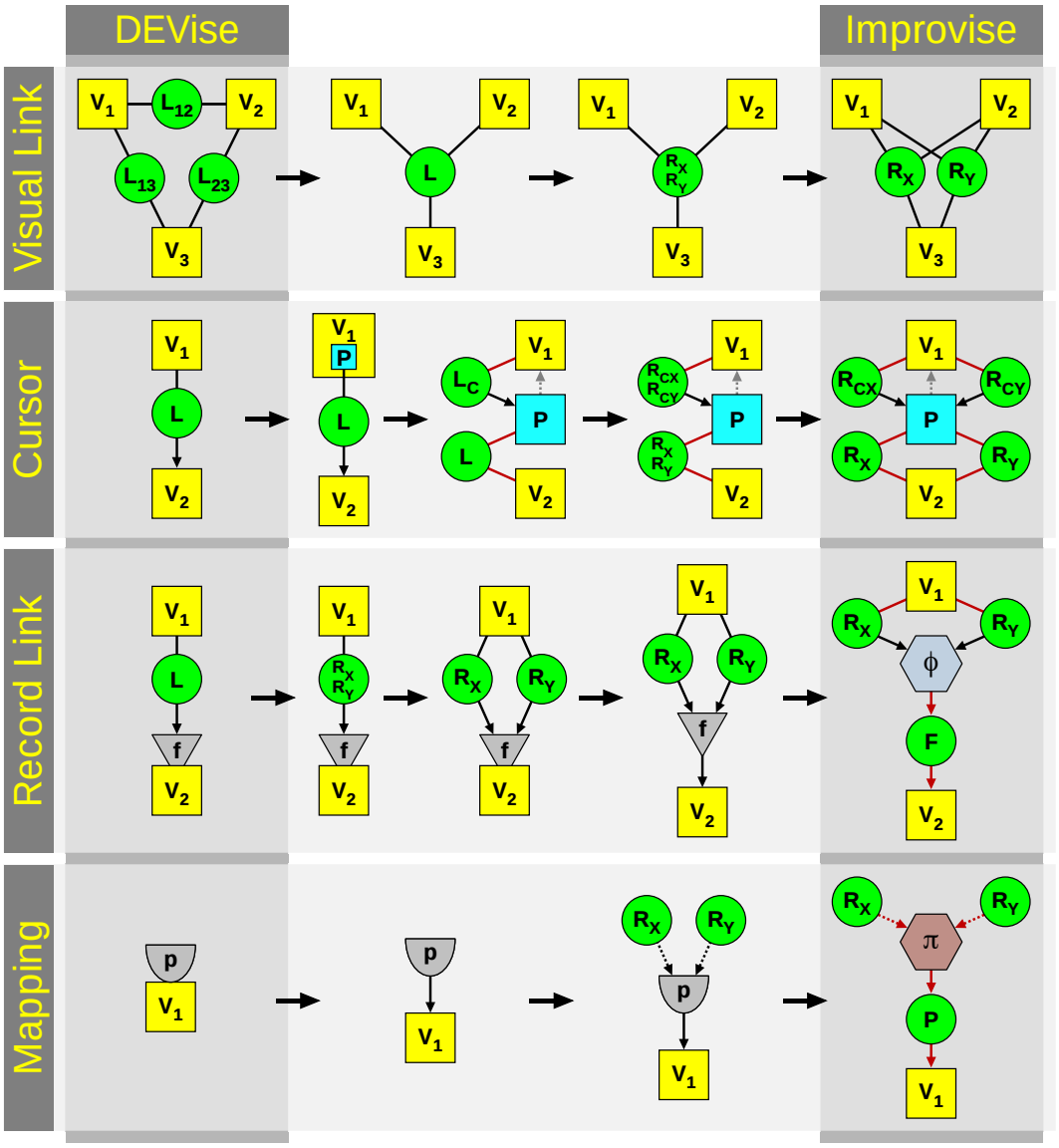

Figure 10: Translating DEVise coordination and visual abstraction primitives into Improvise.

4. The filter can be separated from its view. As a result, filters can be reused by multiple views.

5. In Improvise, views filter records using lexical filter expressions (ϕ) that can be defined in terms of the ranges bound to other views.

Translation of mappings into Coordinated Queries can be broken down into four stages:

1. In DEVise, views (V_1) apply mapping projections (p) to visually encode raw data attributes into graphical attributes of shapes.

2. The projection can be separated from its view. As a result, projections can be reused by multiple views.

3. DEVise uses implicit *visual filtering* to render only projected shapes that fall inside a view's rectangular bounds. The rectangular extent of V_1 can be defined as a pair of ranges on X and Y ($R_X \times R_Y$).

4. In Improvise, views render records using lexical projection expressions (π). Although Improvise scatter plots do not automatically filter unseen glyphs, explicit visual filtering can be done by defining the projection expression so as to make invisible all glyphs that fall outside the view's own ranges.

Translation of record links and mappings reveals two things. First, the manner in which scatter plots filter and visually encode data can be specified externally, whether of not processing occurs internally. Second, these specifications can be defined in terms of ranges that determine the visible extent of the scatter plots. These revelations have several interrelated implications that motivated the design choices behind Live Properties and Coordinated Queries:

- Filters and visual encodings are merely special cases of language fragments that specify data processing operations.
- Data and data processing operations can be shared by multiple views.
- As sharable objects, data and data processing operations are themselves interactive parameters that may depend on other interactive parameters (including other data processing operations).
- Interactive parameters can be of any type, not just ranges.
- Coordination can involve multiple dimensions of different types.
- Coordination can occur between views of any type, not just scatter plots.

Together, these observations suggest that coordination models can be based on *symmetric sharing of interactive parameters between multiple views* rather than on *unidirectional and bidirectional interactive dependencies between pairs of views*. Moreover, interaction in multiple coordinated views can be thought of as *transitions between graphical states in a high-dimensional space of interactive parameters*, thereby reinforcing Chi's contention that Data State Models and Data Flow Models are functionally equivalent ways of constructing CMV visualizations [3].

Using Improvise to metavisualize DEVise coordination structure has resulted in an enormous improvement in speed, usability, and usefulness over earlier approaches. Whereas the original custom-coded metavisualizations took weeks to implement, building the *tgraph* and *devise-mv* visualizations each took hours. Moreover, each of these visualizations provides a complete exploratory interface using multiple coordinated views. These views were added quickly in the Improvise builder interface, but would have been costly in time and effort to implement as custom code. Most importantly, both visualizations have improved understanding of interaction in DEVise by enabling rapid exploration of coordination structure with the goal of identifying and classifying key substructures.

Visual links, cursors, and record links can all be reproduced using synchronized scrolling, overview+detail, and navigation-dependent filtering coordination patterns. Because DEVise coordination structures can be translated easily into Improvise coordination structures, it would be straightforward to recreate all DEVise visualizations in Improvise. The resulting visualizations might then be redesigned or extended with the goal of improving their usability and usefulness by taking advantage of the enhanced coordination and visual abstraction flexibility provided by Live Properties and Coordinated Queries.

Conversely, some simple Improvise coordination structures could be implemented as new DEVise primitives, such as one-dimensional cursors and dimensionally-crossed visual links. An optional interface might be added to DEVise to enable advanced users to create additional, custom coordinations using Coordinated Queries directly inside DEVise. This approach could be followed in other visualization systems as well. Similarly, Improvise might incorporate user interfaces of other visualization systems in the form of abstraction layers that would allow visualization designers and users to interact with their preferred visualization architecture while actually manipulating Improvise views and coordinations.

7 Conclusion

DEVise uses a relational data model to coordinate multiple views of large datasets. Users can create, destroy, coordinate, and specify the contents of scatter plots interactively using a small number of elegant coordinations. However, reproducing common visualization constructions in DEVise frequently involves convoluted chains of linked views, some of which are undesirable artifacts that must be intentionally hidden offscreen.

Two different Improvise visualizations employ standalone metavisualization techniques to support exploration of coordination and screen layout in DEVise visualizations. Both metavisualizations display multiple data sets that represent the statically-captured interactive structure of several DEVise visualizations, but do so at different levels of detail. Coordination graphs reveal that all four DEVise link types can be reproduced by treating the `X` and `Y` ranges of scatter plots as shared objects or as dynamic parameters in simple query expressions. Moreover, exploration and analysis of these graphs uncovered useful variations of the four link types that were later implemented in DEVise. These discoveries suggested a more general approach to coordination that led to the design and implementation of Live Properties and Coordinated Queries in Improvise.

Coordination in DEVise consists of a few well-known patterns similar to those supported in other CMV systems. Although the details of these patterns vary from system to system, all coordinated multiview visualizations share

the same general structure: a graph of views and coordinations. As such, standalone metavisualization techniques are readily applicable across a variety of multiview coordination architectures. These techniques are also a useful fallback when it is impractical to implement integrated metavisualization in existing visualization systems.

Improvise is available (with the *tgraph* and *devise-mv* visualizations) under the Gnu Public License (GPL) at http://www.personal.psu.edu/cew15/improvise/.

Acknowledgments

The author would like to thank the members of the DEVise research group at the University of Wisconsin–Madison, including Miron Livny, Raghu Ramakrishnan, Kevin Beyer, and Kent Wenger.

References

[1] Alan Borning and Robert Duisberg. Constraint-based tools for building user interfaces. *ACM Transactions on Graphics*, 5(4):345–374, October 1986.

[2] David Andrew Carr. *A Compact Graphical Representation of User Interface Interaction Objects*. PhD thesis, University of Maryland, 1995.

[3] Ed H. Chi. Expressiveness of the data flow and data state models in visualization systems. In *Proceedings of Advanced Visual Interfaces 2001*, pages 375–378, Trento, Italy, May 2002. ACM Press.

[4] Jean-Daniel Fekete. The infovis toolkit. In *Proceedings of the IEEE Symposium on Information Visualization 2004*, pages 167–174, Austin, TX, Oct 2004. IEEE.

[5] Allan S. Jacobson, Andrew L. Berkin, and Martin N. Orton. LinkWinds: Interactive scientific data analysis and visualization. *Communications of the ACM*, 37(4):43–52, April 1994.

[6] Eser Kandogan and Ben Shneiderman. Elastic windows: Evaluation of multi-window operations. In *Proceedings of CHI '97*, pages 250–257, Atlanta, GA, March 1997. ACM.

[7] Ravi Krishnamurthy and Moshé Zloof. RBE: Rendering by example. In *Proceedings of the 11th International Conference on Data Engineering*, pages 288–297, Taipei, Taiwan, March 1995.

[8] Shilpa Lawande. A meta-visualization framework for DEVise. Master's thesis, University of Wisconsin–Madison, December 1997.

[9] Miron Livny, Raghu Ramakrishnan, Kevin Beyer, G. Chen, Donko Donjerkovic, Shilpa Lawande, Jussi Myllymaki, and Kent Wenger. DEVise: Integrated querying and visualization of large datasets. In *Proceedings of SIGMOD '97*, pages 301–312, Tucson, AZ, 1997. ACM.

[10] Brad A. Myers. *Creating User Interfaces by Demonstration*. Academic Press, San Diego, CA, 1988.

[11] Chris North, Nathan Conklin, and Varun Saini. Visualization schemas for flexible information visualization. In *Proceedings of the IEEE Symposium on Information Visualization 2002*, pages 15–22, Boston, MA, Oct 2002. IEEE.

[12] Chris North and Ben Shneiderman. Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *AVI '00: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 128–135, New York, NY, USA, May 2000. ACM Press.

[13] Christopher Loy North. *A User Interface for Coordinating Visualization Based On Relational Schemata: Snap-Together Visualization*. PhD thesis, University of Maryland, 2000.

[14] Chris Stolte, Diang Tang, and Pat Hanrahan. Multiscale visualization using data cubes. In *Proceedings of the IEEE Symposium on Information Visualization 2002*, pages 7–14, Boston, MA, October 2002. IEEE.

[15] Craig Upson, Thomas Faulhaber, Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. The application visualization system: A computational environment for scientific visualization. *Computer Graphics and Applications*, 9(4):30–42, July 1989.

[16] Chris Weaver. Building highly-coordinated visualizations in Improvise. In *Proceedings of the IEEE Symposium on Information Visualization 2004*, pages 159–166, Austin, TX, October 2004. IEEE.

[17] Chris Weaver. Visualizing coordination in situ. In *Proceedings of the IEEE Symposium on Information Visualization 2005*, pages 165–172, Minneapolis, MN, October 2005. IEEE.

[18] Allison Woodruff, Alan Su, Michael Stonebraker, Caroline Paxson, Jolly Chen, Alexander Aiken, Peter Wisnovsky, and Cimarron Taylor. Navigation and coordination primitives for multidimensional visual browsers. In S. Spaccapietra and R. Jain, editors, *Proceedings of the 3rd IFIP 2.6 Working Conference on Visual Database Systems*, pages 360–371, Lausanne, Switzerland, March 1995. Chapman & Hall.