Visualization Schemas for Flexible Information Visualization

Chris North, Nathan Conklin, Varun Saini
Center for Human Computer Interaction
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
north@cs.vt.edu
http://infovis.cs.vt.edu/

Abstract

Relational databases provide significant flexibility to organize, store, and manipulate an infinite variety of complex data collections. This flexibility is enabled by the concept of relational data schemas, which allow data owners to easily design custom databases according to their unique needs. However, user interfaces and information visualizations for accessing and utilizing databases have not kept pace with this level of flexibility. This paper introduces the concept of Visualization Schemas, based on the Snap-Together Visualization model, which are analogous to relational data schemas. Visualization schemas enable users to rapidly construct customized multiple-view visualizations for databases in a similarly flexible fashion without programming. Since the design of appropriate visualizations for a given database depends on the data schema, visualization schemas are a natural analogy to the data schema concept.

1 Introduction

Advances in database technology have enabled the widespread collection and proliferation of data. A major contributor to this success is the advent of relational databases, perhaps the most popular storage platform, and its concept of relational data schemas. Data schemas enable data owners to design and define custom database instances that satisfy their unique needs. They do not need to program a new database system, but can simply use data schema tools to organize and manipulate a custom structure for their particular data set. This high level of flexibility supports the definition of an infinite variety of databases, and clearly has had great positive impact on the storage of vast quantities of information.

However, there has not been a similar level of flexibility for constructing effective user interfaces and information visualizations for databases. The design of an appropriate visualization for a given database depends greatly on the data schema and user tasks. Because each data schema is unique, each database requires a unique visualization. General-purpose visualization tools (such as Spotfire [AW95]) can be applied, but often are only a partial solution. For non-trivial data schemas and tasks, appropriate visualizations must be custom programmed. This is an expensive and time consuming effort, even when general-purpose visualizations are utilized within the solution. As a result, many databases do not have adequate visualizations, and data is vastly underutilized.

Furthermore, the flexibility of data schemas enables frequent database modifications. In rapidly evolving data-intensive environments (e.g. bioinformatics), data schemas and domain tasks are in constant flux. As a result, visualizations developed for a specific database are often obsolete by the time they are implemented. Developers are forced to redesign and re-implement, but often cannot keep pace with the rate of change. The problem is exacerbated by the fact that different users and tasks often require different visualizations for the same database.

As shown in Figure 1, this problem represents a fundamental mismatch in design capability between databases and visualization. As an analogy, consider the case if each data schema instance required the data owner to implement a custom database system to store the data.

	Relational Databases	**Traditional Visualization**
Design goal	Data design	Visualization design
Design method	Data schema	Program code
Designer	Data owner	Programmer only
Adaptability	Flexible	Brittle
Rate of change	Rapid, dynamic	Slow, static

Figure 1: Mismatch in design capabilities.

As a result, information visualization researchers and developers often face a common dilemma: At one end of the spectrum, specialized visualizations are very effective for specific targeted data schemas and tasks, but are not broadly applicable to different situations. At the other end of the spectrum, general visualizations can be designed for a broad class of data schemas and tasks (e.g. a scatterplot correlates any 2 numerical attributes), but are not fully adequate for the detailed nuances of individual situations (e.g. 2 attributes that measure health demographics and relate to geography and DNA sequences).

This paper introduces the concept of *visualization schemas* as a solution to this problem. Visualization schemas are based on the Snap-Together Visualization (Snap) model [NS00], and enable database owners or users to rapidly construct custom multiple-view visualizations for databases without programming. Visualization schemas provide an analogy to relational data schemas, and enable a flexible and coordinated approach to the design of data and the design of data visualization.

2 Related Work

Research on flexible construction of visualizations for databases has focused primarily on visualizing single relations (e.g. a single table or query result). Tools such as APT [Mac86], Sage/SageBrush [RCK97], DEVise [LRB97], DataSplash [ACS96], and Spotfire [AW95] enable users to construct visualizations of a single relation by mapping data tuples to visual marks, and then mapping tuple attributes to visual properties of the marks (e.g. x, y, size, color). APT and Sage use an automated approach. DEVise, DataSplash and Spotfire use form-based dialog boxes to let users match data attributes to visual properties. SageBrush most closely resembles visualization schemas, using a visual language and user interface similar to a paintbrush program. Many of these systems enable the simultaneous display of multiple such visualizations of the relation, linked for brushing [BC87]. DEVise and DataSplash enable users to link the visualizations for synchronized pan and zoom. [PCS95] provides a visual specification for such links. In LinkWinds [JBO94], users can link views into a pipeline for filtering data.

For databases containing multiple relations, Visage/VQE [DRK97] extends attribute mapping, brushing, and dynamic queries to multiple relations. Users can perform the operations on tuples and attributes in different relations that are associated by a join. DataSplash lets users construct 'wormholes' in a semantic zooming space that lets them drill down across relations by zooming in. RMM [ISB95] uses a *hypertext schema* approach, similar to visualization schemas, based on entity-relationship diagrams to construct data-driven websites. Tuples map to web pages, relations map to index pages, and associations map to hyperlinks.

Dataflow systems such as AVS [UFK89] and GeoVista [TG02] enable flexible specification of data flow networks. Because of the focus on computationally intensive scientific-visualization applications, dataflow diagrams have evolved more as a representation for data processing rather than a specification for visualizations or user interfaces (perhaps away from Haeberli's original vision with ConMan [Hae88]). Due to the complexity of such environments, dataflow systems are generally geared towards programmers as users.

There are many flexible visualization toolkits and frameworks, such as Rivet [BST00] and Sieve [IBW97], that can produce similar multiple-view visualizations as visualization schemas, but require programming.

3 Database Background

Relational databases consist of three major layers: The *relational data model* provides compositional principles to organize information in a structured form that is usable and manipulable. *Relational data schemas* enable users to define a custom structure for a particular dataset. *Relational database management systems* (DBMS) implement the model and schema. Together, these layers enable users to organize, store, manipulate, and share data in a flexible and customized fashion.

From the users' perspective, the design flexibility is accomplished through the use of data schemas. Relational data schemas provide several primitives that users compose to create a database: attributes, tuples, and relations. Many modern database systems also explicitly represent join associations (primary- and foreign-keys).

Many database systems provide a visual language for data schemas in the form of a diagrammatic, direct manipulation user interface. Diagrammatic data schemas are generally represented as a graph. Nodes represent relations, and each node shows a list of its attributes. Edges represent join associations. Directed edges represent one-to-many associations, and undirected edges represent one-to-one associations.

For example, Figure 2 shows the data schema for a database of website hits in Microsoft Access. The database contains information about the website's pages ("URLs"), hits to pages, external referring pages that link to the website's pages, etc. A one-to-many association exists between pages and hits.

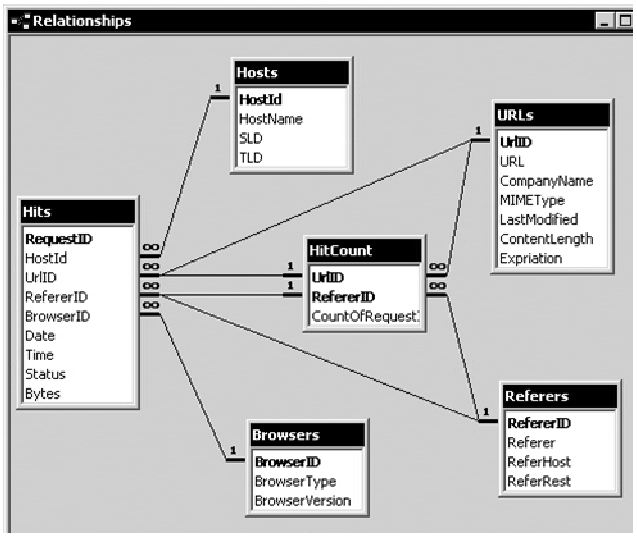


Figure 2: Diagrammatic data schema for a database of website hits in Microsoft Access.

Data schemas, especially diagrammatic data schemas, have many important benefits:

- Enables data owners to define and modify the structure of a database using a simple language.
- Provides guidance for data design, and enforces rules of the data model (e.g. validity).
- Provides an overview of database structure to help other users understand database contents.
- Provides usability for data storage tasks.
- Enables systems to store and interpret database contents interchangeably.

Visualization schemas can have similar benefits for visualization.

4 Snap-Together Visualization

Visualization schemas are based on Snap-Together Visualization (Snap). The primary goal of Snap is to enable a level of flexibility in visualization design that rivals that of data design. To accomplish this, the primary guiding principle of Snap is to establish a tight analogy between relational database concepts and visualization concepts (summarized in Figure 3). Snap consists of three visualization layers that are analogous to and extend the three relational database layers: The *Snap visualization model* extends principles of the relational data model to the visualization realm (reported in [NS00]). *Snap visualization schemas* are analogous to relational data schemas, enabling users to define custom visualizations for databases. The *Snap visualization server* operates on top of a relational DBMS.

The Snap visualization model is based on composition of multiple-views. In the multiple-view visualization approach, data is displayed in several different views (called *visualization components*). Different components may display the same or different portions of the data. These components are then tightly coupled (or *coordinated*) in a variety of ways [Nor01] such that interacting with one component causes meaningful effects in others. This produces an integrated composite or *multiple-view* visualization [BWK00].

Snap's basic assumption is that the fundamental unit of creative design in information visualization is an individual visualization component. A secondary goal of Snap is to leverage component implementations from the field. Our hope is that this will create a pipeline that transfers tools from research to practical application, and thus facilitates information visualization in "crossing the chasm" [Shn01]. Visualization components are donated by their inventors; coordinations are constructed by users.

5 Visualization Schemas

Visualization schemas establish a direct correspondence between data schema primitives and visualization schema primitives:

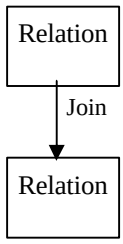
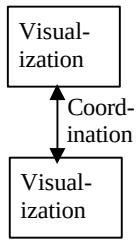
	Relational Databases	**Snap Visualization**
Design goal	Data design	Visualization design
Design method	Data schema	Visualization schema
Designer	Data owner	Data owner
Adaptability	Flexible	Flexible
Rate of change	Rapid, dynamic	Rapid, dynamic
Layers:		
Theory	Rel. data model	Snap vis. model
User interface	Rel. data schema	Snap vis. schema
Architecture	Rel. DBMS	Snap vis. server
Schema Primitives:		
Conceptual	Relation	Vis. component
	Tuple	Visual item
	Attribute	Visual property
	Join association	Coordination
Diagrammatic		

Figure 3: Analogy between relational database concepts and visualization concepts enables a matching level of design capability.

- *Visualization component* = relation. A visualization component is a view that displays a data relation or query result (we assume queries are integrated into the data schema as relations). A visualization component can be a hard-coded implementation or dynamically generated using the attribute-mapping technique. An example is a scatterplot component that displays a binary relation (or a binary projection of a relation with larger arity).
- *Visual item* = tuple. Tuples are displayed as visual items in a visualization component. For example, a tuple is displayed as a dot in the scatterplot. User interactions in visualization components are tuple based. For example, users might select a set of items or zoom onto an item.
- *Visual property* = attribute. Attributes are used by visualization components to compute graphics. Users map data attributes to component-specific visual properties. For example, an attribute maps to the x axis on the plot.
- *Coordination* = join association. A coordination between two visualization components tightly couples them according to the join between their relations. User interactions on tuples in one component causes visual actions on associated tuples in the other component. Users choose the component-specific interactions to be tightly coupled. For example, a brushing coordination between two scatterplots of the same relation tightly couples the

'highlight' actions across the implicit one-to-one join. Then, when users highlight items in one plot, the associated items are automatically highlighted in the other plot. See [Nor01] for a taxonomy of other coordinations.

Visualization schema diagrams are represented visually similar to data schema diagrams. Visualization schemas are represented as a graph. Nodes represent instantiated visualization components. Nodes indicate the visualization type, including a miniature icon of the visualization, and the data relation displayed in the visualization. The actual visualization that the node represents is displayed in the visualization workspace in a separate frame elsewhere on the screen. Edges represent coordinations between visualizations. Edges indicate the actions that are tightly coupled on each side of the coordination, the join association keys, and the join type (one-to-one or one-to-many).

5.1 Example

With the above database of website hits, we construct a visualization of the website, page popularity, and referring pages. Figure 4 shows the constructed visualization workspace, and Figure 5 shows its visualization schema. First, we construct a simple website browser with visualization of page popularity (top 3 components in both Figures).

To design the visualization, dragging the URLs relation into the visualization schema, and selecting a standard tree-view visualization from a menu, displays the top-left node in the schema. In the visualization workspace, the tree structure of the website is displayed in the tree-view visualization component (top-left). Similarly, the URLs relation is again dragged and displayed in a Treemap visualization component (top-center). The number of hits to each page (popularity) is mapped to rectangle size and color in the Treemap.

In the visualization schema, dragging a link from the tree-view to the Treemap establishes a coordination. The 'select' actions are chosen on both ends for tight coupling to enact a brushing type of coordination. The implicit one-to-one join between URLs and URLs is automatically chosen. Now, in the visualization workspace, selecting web pages in either the tree-view or Treemap also selects and highlights the pages in the other. Users can browse from either perspective.

The URLs relation is dragged and displayed in a web-browser component (top-right). A link is dragged from the Treemap to the web-browser icons, and the 'select' and 'load' actions are chosen respectively. Now, selecting a page in the Treemap or tree-view will load that URL into the web-browser and display the actual web page. The designed visualization now enables users to browse the website from the tree-view and popularity Treemap perspectives and view desired pages.

We then add the capability to visualize referring pages (bottom 2 components in both Figures). The scatterplot and web-browser components at the bottom display data about external web pages that refer readers to the site. Selecting a page in the tree-view or Treemap shows the page in the top web-browser, and displays referrers to that page in the scatterplot. The plot displays referrers alphabetically on the x-axis and by number of referred hits on the y-axis. Selecting a referrer in the plot displays the referring page in the bottom web-browser. The example shows that most readers discover the website's homepage through Google.

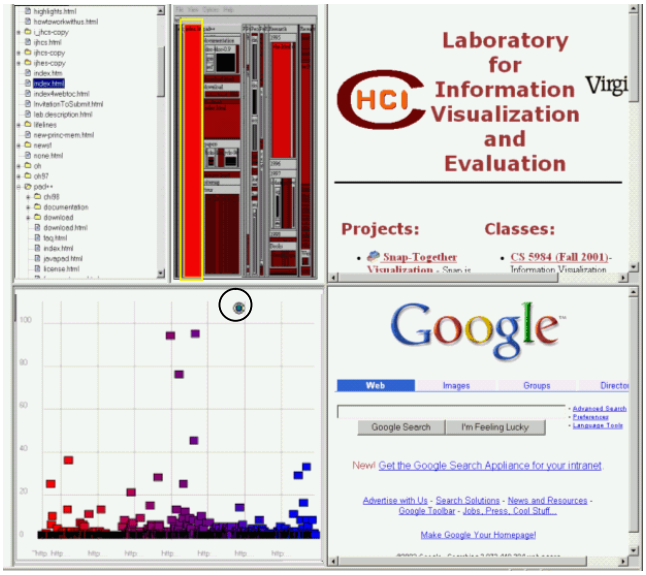


Figure 4: Constructed visualization workspace for website hits database.

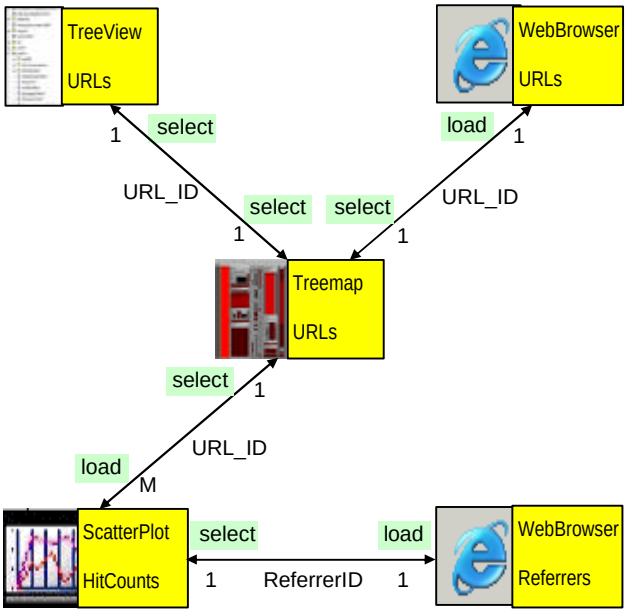


Figure 5: Visualization schema for Figure 4.

5.2 Interaction

Opening a new visualization component adds a node to the graph. Users can drag nodes to rearrange the graph. Users can change the visualization type of a node by selecting a new visualization component from the menu. For example, if the user initially displays a relation in a scatterplot, but then decides that a parallel-coordinates plot would better show the data, simply selecting the parallel-coordinates item will close the scatterplot and open the parallel coordinates plot in its place. Users can load and display a data relation into a component by dragging the relation from the data schema to the component's node. This can be repeated to display different data in the component. Furthermore, users can directly select which attributes of the relation to include in the component. This provides a quick and simple form of projection query that is common in visualization tasks.

Some visualization components require data attributes to be matched to visualization properties at load time. For example, a scatterplot component might have x-axis and y-axis properties. The visualization schema interface enables users to specify these mappings by selecting data attributes for each property.

Dragging a link from one node to another establishes a coordination between the visualization components. Users select the user interface actions to coordinate from drop down menus on each end point of the edge. Each visualization component has its own set of coordinate-able user interface actions. Each action has a cardinality of either one or many. For example, a select action in some components might allow only one item to be selected at a time, whereas other components may support multiple selection. If a component is on the 'many' side of a coordinated join, then single item actions are grayed to indicate an inappropriate match. Once actions are selected, the actual visualization instances will then behave in the specified coordinated fashion.

When coordinating, if there are multiple possible join paths between the visualization components' relations then the user can select the desired join path from a drop-down list. The join path can be a direct association between the two relations, or an indirect association that concatenates multiple joins through intermediate relations. The system uses the data schema to determine possible join paths, and automatically computes the necessary join query. Visualization components and coordinations can be eliminated using right-click menus.

5.3 Advantages

Visualization schemas have many advantages. Its direct manipulation approach enables users to construct and modify visualizations very rapidly. Because it closely resembles relational data schemas, its learning time is greatly reduced from the previous form-based approach [NS00] and it helps users learn the Snap model concepts. Also, visualization schemas specify user interactions, which is a level that users are familiar with, in contrast to dataflow specifications which are at a much more detailed data processing level.

Furthermore, it provides a visual overview of the coordination structure of the visualization which helps users to quickly learn how to operate the visualizations and understand its response. This helps to solve a long-standing usability problem with multiple-view visualizations in general. Hence, visualization schemas improve usability of both the construction phase as well as the operation phase. Visualization schemas also can provide a compact representation of visualizations for the purpose of browsing and recognizing many alternate designs. It also provides a context in which to show other information related to visualizations and their use, such as animations of propagation of actions through the coordination graph or saved sequences of actions such as macros.

Finally, it enables a new level of flexibility in visualization that matches that of databases, enabling users to not only store data but visualize it as well. It does not require programming, which enables visualizations to be updated along with dynamic database schemas.

6 Datacompass

The DataCompass is an alternate form of visualization schema intended for novice users or very complex database schemas. Visualization schemas employ a two step process for construction. Users first display data in visualization components, then coordinate the components. This approach requires users to have some forethought about their visualization design goal and familiarity with the data schema. While this may be appropriate for data-savvy users, novices need more guidance especially when using databases with many relations and complex schemas.

To this end, DataCompass simplifies construction to a single step process. Starting with an initial relation displayed in a visualization component, DataCompass enables users to navigate to associated relations by simply selecting it from a compass-like layout. The associated relation opens in another visualization component and automatically coordinated to the previous component. Hence, it answers the question "what data can I navigate to from here?", and provides a fast mechanism for coordinating to the chosen relation.

In DataCompass, the current visualization component and relation is shown highlighted in the center. This is the component that the user most recently interacted with, or invoked DataCompass from. The relations that the user can coordinate to the current component are placed in the DataCompass based on their join association to the current component's relation. The relationships are automatically determined from the data schema. The relations are divided into three groups according to hierarchical associations: (Figure 6)

- One-to-many towards the current relation: displayed on the north of the compass.
- One-to-one: displayed on the east and west.
- One-to-many towards the other relation: displayed on the south.

The DataCompass (Figure 7) initially shows only directly associated relations for simplicity, but additional levels of associations are available by mouse-over. If a relation has multiple join paths to the current relation, it is shown in multiple places and labels indicate relevant keys.

Users can then select any of the relations on the DataCompass to visualize and coordinate it to the current visualization component. When selecting a relation, users are prompted to choose a visualization type and the actions to coordinate. The relation is then immediately displayed and coordinated.

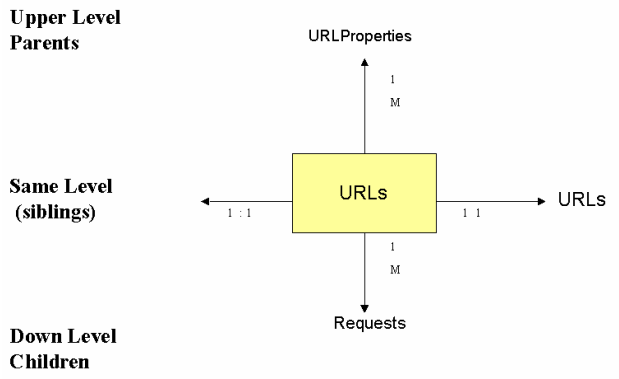


Figure 6: DataCompass conceptual design. In this example, a component displaying the URLs relation is the focus. DataCompass indicates that the user can now instantiate and coordinate to URL Properties, Requests, or URLs again.

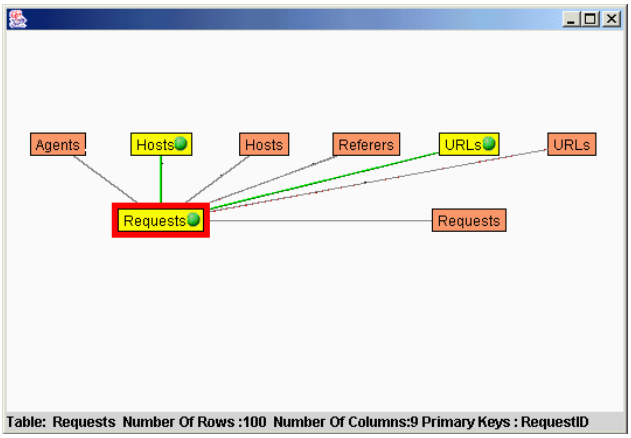


Figure 7: DataCompass implementation. In this example, the Requests relation has been shown in a component and is currently coordinated to Hosts and URLs relations. Potential new coordinatations are available to parent and sibling relations.

In the DataCompass, yellow nodes represent relations that are already displayed in visualization components. Red nodes represent relations in the data schema. Green edges represent existing coordinations between instantiated components. Gray edges represent potential coordinations to data relations or instantiated components. This enables users to coordinate to existing instantiated components, alter existing coordinations, or instantiate and coordinate new components.

If a relation is displayed in multiple components, then it will have multiple nodes in DataCompass. Also, there is always a red node for each relation, even if the relation has already been displayed in a component. The focus relation always shows a sibling one-to-one association with itself. This enables users to instantiate multiple components for a relation, perhaps for a brushing style coordination.

Visualization Schemas and DataCompass are interchangeable. Using DataCompass builds up the visualization schema. At any time, users can switch between the two.

Visualization Schemas and DataCompass are two different approaches that match two different user tasks in Snap. Visualization schemas are designed for a *top down* approach in which users have a partially defined idea of their desired visualization design. They construct a visualization so that they (or others) can explore the data. This is similar to the goal-oriented approach taken when structuring data using data schemas. Typically, these users are knowledgeable about relational database concepts and the data schema. Whereas, DataCompass is designed for a *bottom up* approach that is more exploratory in nature. Users are novices or unfamiliar with the data schema, and construct a visualization workspace as a result of navigating and exploring the data.

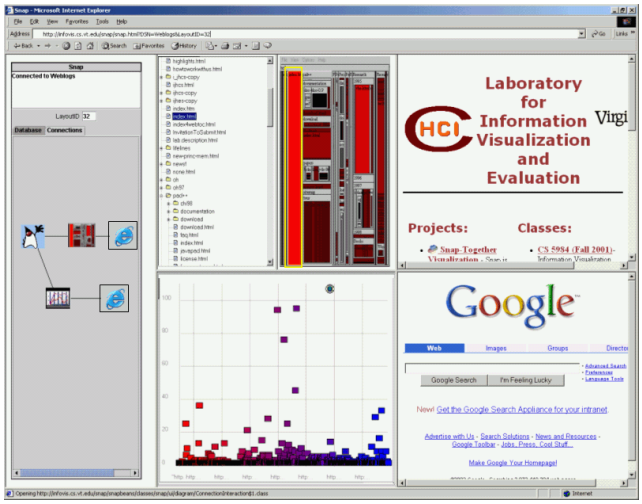


Figure 8: Visualization schemas are shown adjacent to the visualization workspace.

7 Snap Visualization Server

The Snap visualization server, analogous to the relational DBMS, is the software system that implements visualization schemas. The server is a web-based system that allows users to construct multiple-view visualizations of any accessible relational database. Snap server runs in a web browser and uses frames to organize multiple visualization components in a tiled space-filling fashion (Figure 8). This helps reduce window management problems. Users can add components by splitting frames horizontally or vertically. The visualization schema is tiled on the left side.

Users can save the visualization workspace and schema that they have constructed. The visualization can then be re-opened with a single URL. This allows users to easily publish and share their data and visualizations. Snap can also save each visualization component's state. Using event playback, Snap can reestablish each visualization component to its previously saved state. This technique provides for asynchronous collaboration, allowing users to share the insight that they gained while interacting with visualizations.

With Snap, users can connect to databases stored on their local client machine, on the Snap server, or on remote 3rd party database servers such as the Census Bureau (Figure 9).

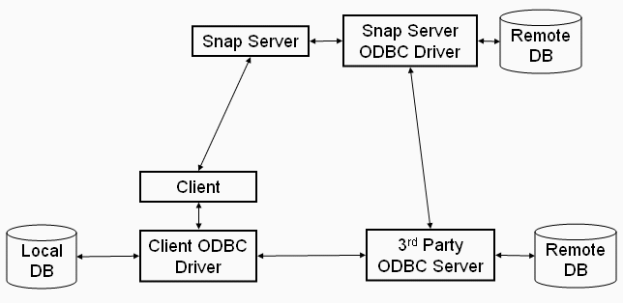


Figure 9: Data schema access points.

Snap also leverages 3rd party visualization components. Developers of visualization components can submit their components to the Snap server, enabling other users to utilize the components in their visualization schemas. Hence, the Snap server acts as a repository for visualization components, and facilitates the transfer of new components to practical application.

Snap server is implemented in Java. The software model is an event-based, implicit-invocation architecture. This allows component developers to be concerned only with firing and receiving events and not with explicitly coordinating with other components [SG96]. To make their components snap-able, developers must implement our *snapable* programming interface. This interface is designed to be very simple and to minimize developers required effort. The three methods in the interface require that the component be able to load data given to it from Snap, send action events to Snap, and receive action events from Snap. The server is runtime extensible, meaning that developers can submit components without the need to re-compile or restart the server.

The architecture also has an adapter layer that enables visualization components implemented in many different technologies to be coordinated in visualization schemas (Figure 10). Hence, visualization components can be Javabeans, Java applets, ActiveX components, and Javascript pages.

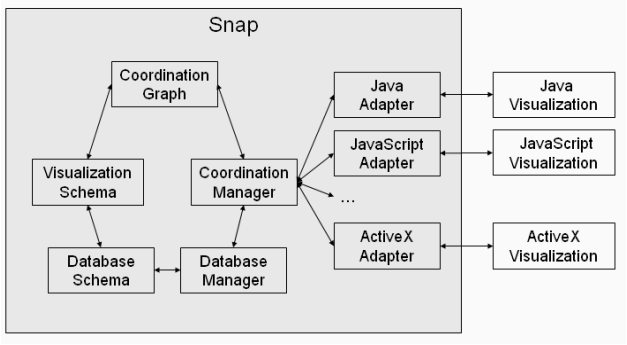


Figure 10: Snap server architecture.

8 Conclusions and Future Work

The primary contributions of this work are:

- The concept of Visualization Schemas as a natural extension to data schemas that enable a corresponding level of flexibility for constructing information visualizations as for storing data.
- The DataCompass variation that accommodates data-novice users and very complex data schemas.
- The Snap universal visualization server that enables collaboration and extensibility so that visualization schemas can leverage new visualization developments from the field

Our future vision is of the integration of databases and interfaces ("Datafaces"), by integrating visualization schemas with data schemas. This will enable users to simultaneously manipulate the data and visualization, and to see how data and visualization relate. This will help to solve the frequent problems that occur when changes are made to a database that causes visualizations for the database to become obsolete. Datafaces would enable database administrators to recognize the impacts of data schema changes and immediately update visualizations to match. Furthermore, it would help users to modify data schemas for the purpose of generating a desired visualization. It may be possible for Datafaces to highlight invalidated conditions, suggest alternatives, or automatically produce visualizations based on heuristics (e.g. a multiple-view extension to Sage [RCK97]). Such

capability could enable every database to have an appropriate user interface.

Snap is available at: http://infovis.cs.vt.edu/

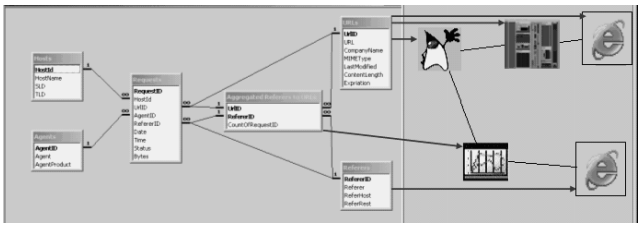


Figure 12: Datafaces concept

9 Acknowledgments

Thanks to Matt Clement, Jeevak Kasarkod, Rohit Kelapur, and Atul Shenoy for development contributions.

10 References

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

[ACS96] Aiken, A., Chen, J., Stonebraker, M., Woodruff, A., "Tioga-2: A Direct Manipulation Database Visualization Environment", *Proc, 12th Intl. Conference on Data Engineering*, pg. 208-217, (February 1996).

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

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

[BST00] Robert Bosch, Chris Stolte, Diane Tang, John Gerth, Mendel Rosenblum, and Pat Hanrahan. "Rivet: A Flexible Environment for Computer Systems Visualization", *Computer Graphics* 34(1), (February 2000).

[DRK97] Derthick, M., Roth, S.F., Kolojejchick, J., "Coordinating declarative queries with a direct manipulation data exploration environment", *IEEE Information Visualization Symposium*, pp. 65 -72, (1997).

[Hae88] Haeberli, P., "ConMan: A Visual Programming Language for Interactive Graphics", *Proc. ACM SigGraph '88*, pp. 103-111, (1988).

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

[IBW97] Philip Isenhour, James "Bo" Begole, Winfield S. Heagy, and Clifford A. Shaffer, "Sieve: A Java-Based Collaborative Visualization Environment," *In IEEE Visualization '97*, (October 1997).

[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).

[Mac86] Mackinlay, J., "Automating the design of graphical presentations of relational information". *ACM Trans. on Graphics* 5(2), 111-141, (1986).

[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).

[NS00] North, C., Shneiderman, B., "Snap-Together Visualization: Can Users Construct and Operate Coordinated Views?", *Intl. Journal of Human Computer Studies*, Academic Press, 53(5), pg. 715-739, (November 2000).

[Nor01] North, C., "Multiple Views and Tight Coupling in Visualization: A Language, Taxonomy, and System", *Proc. CSREA CISST 2001 Workshop on Fundamental Issues in Visualization*, pp. 626-632, (2001).

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

[RCK97] Roth, S., Chuah, M., Kerpedjiev, S., Kolojejchick, J., Lucas, P., "Towards an Information Visualization Workspace: Combining Multiple Means of Expression", *Human-Computer Interaction Journal*, 12(1&2), pp. 131-185, (1997).

[SG96] Shaw, M. and Garlan, D., *Software Architecture: Perspectives on an Emerging Discipline.* Prentice Hall, Inc. (1996).

[Shn01] Shneiderman, B., "Supporting Creativity with Advanced Information-Abundant User Interfaces", in *Human-Centered Computing, Online Communities, and Virtual Environments*, Springer-Verlag, pp. 469-480, (2001).

[TG02] Takatsuka, M. and Gahegan, M "GeoVISTA Studio: A Codeless Visual Programming Environment For Geoscientific Data Analysis and Visualization", *The Journal of Computers & Geosciences*, (2002).

[UFK89] Upson, C, Faulhaber, T, Kamins, D, Laidlaw D, Schlegel D, Vroom J, Gurwitz, R, van Dam, A, "The Application Visualization System: A Computational Environment for Scientific Visualization", *IEEE Computer Graphics and Applications*, pp. 30-42, (July 1989).