DEVise and the JavaScreen: visualization on the web

Hongyu Yao, Kent Wenger, Miron Livny

Department of Materials Science and Engineering (Yao) University of Wisconsin-Madison 1500 Engineering Drive Madison, WI 53706

Computer Sciences Department (Wenger and Livny) University of Wisconsin-Madison 1210 West Dayton St. Madison, WI 53706

{hongyu,wenger,miron}@cs.wisc.edu

ABSTRACT

DEVise (Data Exploration and Visualization) is a general-purpose data visualization and exploration system capable of handling datasets from a variety of sources. Users create visual presentations, which they, or others, can dynamically explore to search for patterns in their data. The JavaScreen is a Java applet we have recently developed, which functions as a DEVise client. With it, users can explore visual presentations via the web, without having to actually install the DEVise software. Our paper covers three main areas: DEVise itself, including the fundamental DEVise model and adaptations required to support the JavaScreen; the JavaScreen and related software; and the use of DEVise and the JavaScreen in several applications.

Keywords: data exploration, data visualization, world wide web, client-server, visual query

1. INTRODUCTION

As users seek to gain insights from an ever-increasing amount of data, data visualization becomes more important, since it allows the use of human pattern-recognition capabilities to discover non-obvious correlations in data sets. DEVise was conceived and developed as a tool to allow users to flexibly visualize and explore data, thus maximizing their ability to discover knowledge implicit in the data.

DEVise was initially implemented as a single-user X Window-based system. However, a number of users want to use DEVise presentations on platforms that do not support the X Window System, or make their DEVise presentations available to multiple users at the same time. Moreover, more and more users want to access DEVise presentations via the world wide web.

To overcome the shortcomings of the original DEVise system while preserving the dynamic nature of DEVise presentations, we developed the JavaScreen, a DEVise multiuser environment that can run on any platform supporting Java. Instead of being limited to the X Window System, users can now view a DEVise presentation on a virtual screen running on a wide variety of platforms, without having to install additional software.

There are a number of other data visualization projects such as DataSplash[1] (formerly Tioga-2, the University of California-Berkeley), Pad and Pad++[2] (the University of Maryland, the University of California-San Diego, the University of New Mexico, the NYU Media Research Laboratory and Bellcore), Jazz[3] (the University of Maryland), SageBrush[4] (Carnegie Mellon University) and Visage[5] (Carnegie Mellon University and the Maya Design Group). Important features of DEVise relative to this other work include the following:

- DEVise presentations are constructed graphically and interactively, without any programming necessary.
- DEVise emphasizes *connections* between views.
- DEVise does not rely on the assumption that all data fits into main memory.
- DEVise presentations are available through a virtual screen such as the world wide web with the support of JavaScreen.

Beyond simply allowing access to DEVise via the world wide web, the JavaScreen and JavaScreen Point of Presence allow a single DEVise server to serve several clients, without fundamental modifications to the DEVise (which can only support a single client at any given time). The JavaScreen also represents a significant change in the interaction of the DEVise client and server.

This paper is organized as follows: Section 2 is a description of DEVise itself, focusing on features of DEVise that are visible at the user level. For the purposes of this paper, we ignore the underlying structure of the DEVise software. Section 3 explains the JavaScreen and JavaScreen Point of Presence, and also discusses changes that were made to the DEVise server to support the JavaScreen. Section 4 discusses two applications of the JavaScreen, and we conclude with sect. 5.

2. DEVISE DESCRIPTION

DEVise is a record-oriented system: in general, one symbol is drawn in a view for each record of the underlying data. The user creates a presentation (or "session") by importing data and creating views in which to display the data, laying out the views (including overlaying them), and making connections between them. A presentation can be saved to a session file for re-use.

Most of the power of DEVise comes from a small number of primitives allowing the user to control the representation of data, and to make connections between various data views. Cursors are the most common primitive for data exploration in DEVise. They allow the user to select a region of data by simply clicking or dragging with the mouse. Often, a combination of cursors in several global views is used to select the data to be shown in a detailed view (examples of this will be discussed in the applications section).

2.1 Data to visualization

The steps from input data to screen image are shown in Figure 1.

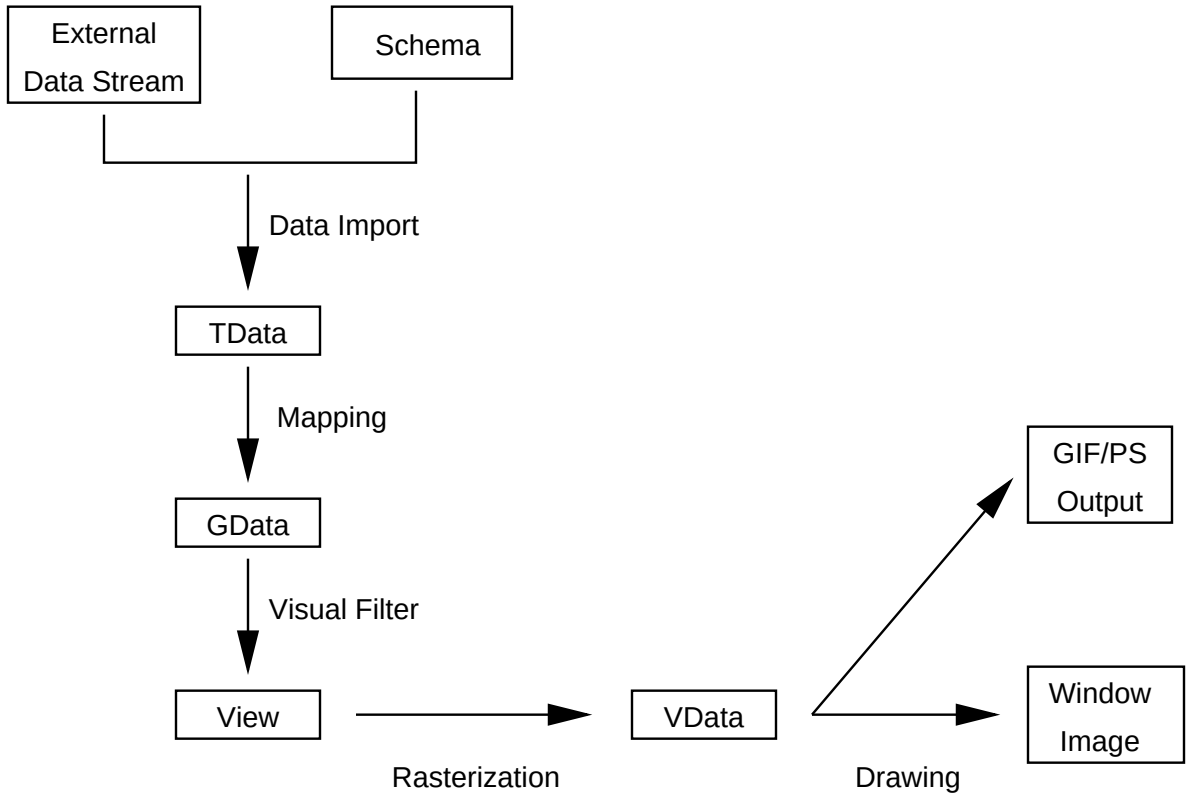

FIGURE 1. DEVise data visualization model.

A DEVise visualization starts with data in a file or other data stream. This data stream is parsed using a schema defined by the user. The schema specifies the physical layout of fields in the data records and assigns meaningful names to the fields. For example, a data file may contain a column of dates and a column of temperatures. The object resulting from the combination of a data file and a schema is known as a TData (tabular data) object.

Once a TData object has been created, it is further processed using a *mapping*. The mapping specifies a transformation from the attributes of the TData to a set of fixed attributes of a GData (graphical data) object. For example, the mapping may specify that the TData's date attribute should be the GData's X value, and the temperature attribute should be the Y value. The mapping also specifies other GData attributes, such as size, color, and symbol type. (Often, some of the GData attributes are specified as constants in the mapping.)

Once a GData object is created, it is inserted into a view. The GData records that will actually produce symbols are selected by the view's visual filter, which is a selection on the GData's X and Y values (in the case of a two-dimensional view). An additional selection takes place if the view is the follower of a record link (see below).

When a view needs to be painted it issues a *visual query* on the appropriate GData (this is a request for GData records that match the view's visual filter). Lower levels of the software return the appropriate records, re-reading the original data file and re-converting from TData to GData if necessary.

A combination of views, the connections between them, and the screen layout is known as a session. DEVise sessions are constructed graphically by the user, without the need for any actual programming. At the present time, sessions can only be constructed with the non-JavaScreen version of DEVise; the JavaScreen can be used to explore sessions, but not create them.

2.2 View connections

There are five types of connections between views in DEVise: visual links, record links, cursors, view symbols, and piles.

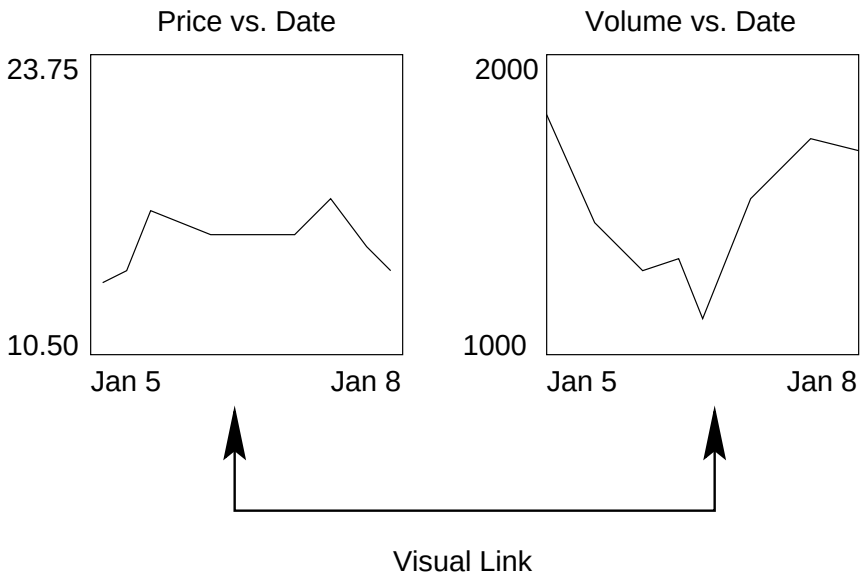

FIGURE 2. Visual link on X axes.

A visual link (see Figure 2) is a connection between the visual filters of two or more views; for example, two views may share the same X axis range. Visual links can be X-only, Y-only, or XY. A visual link is the only type of connection that is symmetrical: changing a linked attribute in any of the linked views changes all of the views. Since visual links only affect the linked views' visual filters, views can be visually linked even if they have different data sources (unlike *record links*, which are discussed below).

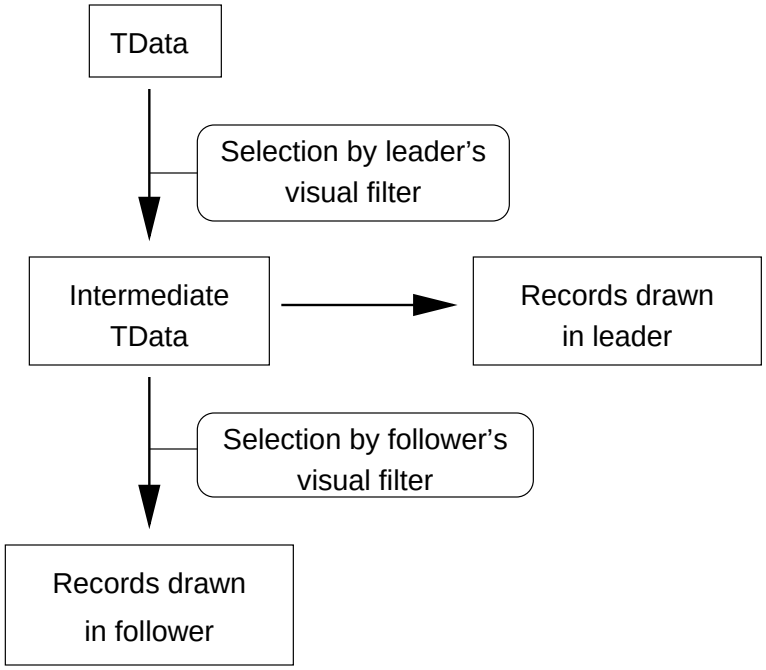

FIGURE 3. Record link.

A record link (see Figure 3) is a connection between a leader view and a follower view such that only records that appear in the leader view can appear in the follower view. (A record link can also be *negative*, in which case only records that do *not* appear in the leader view can appear in the follower view.) Record links can be thought of as applying the visual filters of both the leader and follower views to the data displayed in the follower view.

A record link is unidirectional: the follower view has no effect on the leader view. Record links are only applicable when the leader and follower views display the same TData. If a view is a follower of multiple record links, the set of records that it can display is the intersection of the sets specified by each link.

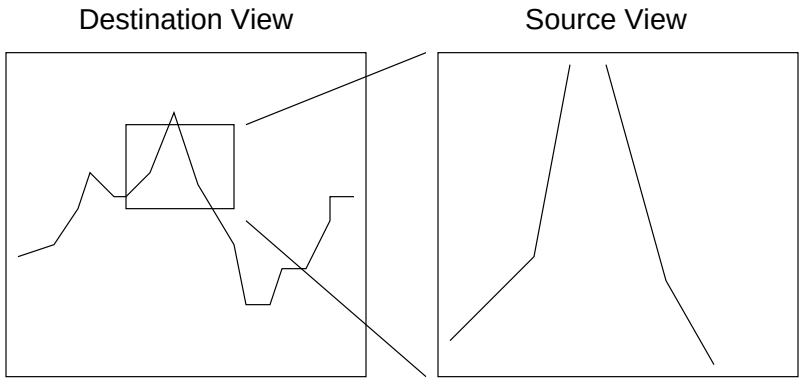

FIGURE 4. Cursor on X and Y axes.

A cursor (see Figure 4) is a connection between the visual filter of one view (the source view) and a draggable rectangle displayed in another view (the destination view). A cursor is bidirectional but not symmetrical. Moving the cursor in the destination view changes the visual filter of the source view, while directly changing the source view's visual filter moves the cursor

in the destination view. Like a visual link, a cursor can be X-only, Y-only or XY; and the source and destination views can display different TDatas.

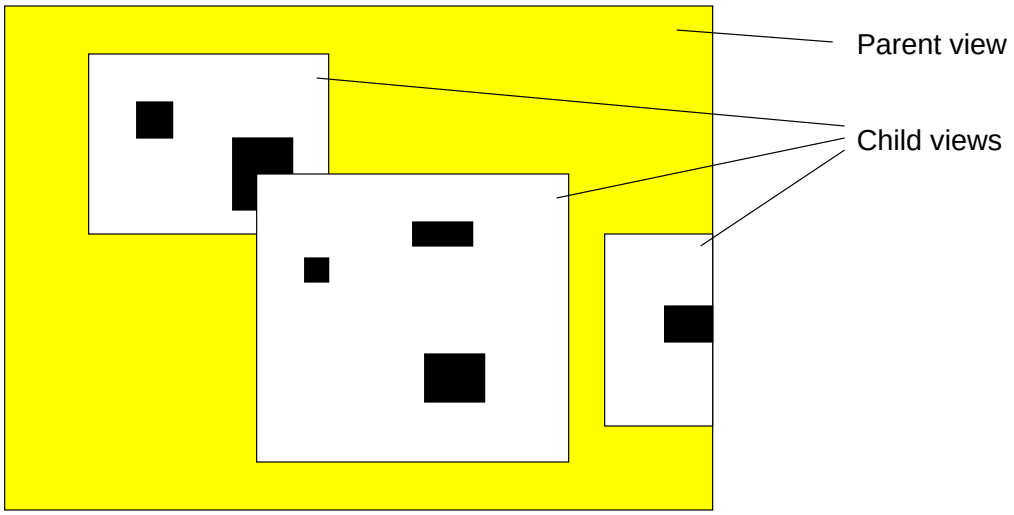

FIGURE 5. View symbols.

A view symbol (see Figure 5) is simply a view (the child view) that is drawn as a symbol with another view (the parent view). From the parent view's perspective, a child view is a symbol like any other symbol, except that it has additional properties that can be controlled. From the child view's perspective, it is a view like any other, except that some of its properties are controlled by its parent. Fundamentally, the parent controls the size and location of the child, as with any other symbol type. Optionally, the parent view can also control which TData the child view displays, and parameters of the child view's mapping. This allows, for example, the combination of a cursor, record link, and view symbol to control which data is seen in a child view. View symbols are a unidirectional connection: the child view has no control over the parent view.

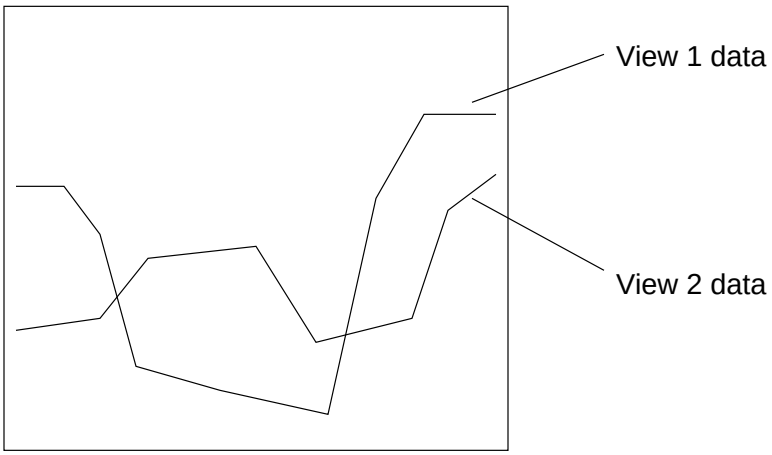

FIGURE 6. Piled views.

A pile (see Figure 6) consists of one or more views occupying the same screen real estate. All but the "bottom" or "back" view are effectively drawn with a transparent background, so only the background and axes (if any) of the bottom view are actually shown. Piles are generally used for overlaying two or more sets of data. Often, all views in a pile are visually linked on their X and Y axes, but this is not always the case. The order of the views in a pile can be changed to put the appropriate data on "top". View symbols can also be piled. In this case, the pile order is determined by the Z value specified in the parent view's mapping.

Piles are different from the other types of view connections in that they do not affect *what* is displayed in a view, only *how* it is displayed.

3. THE JAVASCREEN

DEVise provides a solid framework for data visualization; however, it does have a few limitations. Most of all, it is a single-user X Window-based system that can only run on certain platforms. In order to support the user of DEVise in a more distributed environment, such as the world wide web, we have developed the JavaScreen using the platform-independent Java programming language.

The JavaScreen consists of three main pieces of software: the JavaScreen Applet (JSA), the JavaScreen Server (JSS) and the JavaScreen Point-of-Presence (JSPop). The relationship between them and the DEVise server is shown in Figure 7.

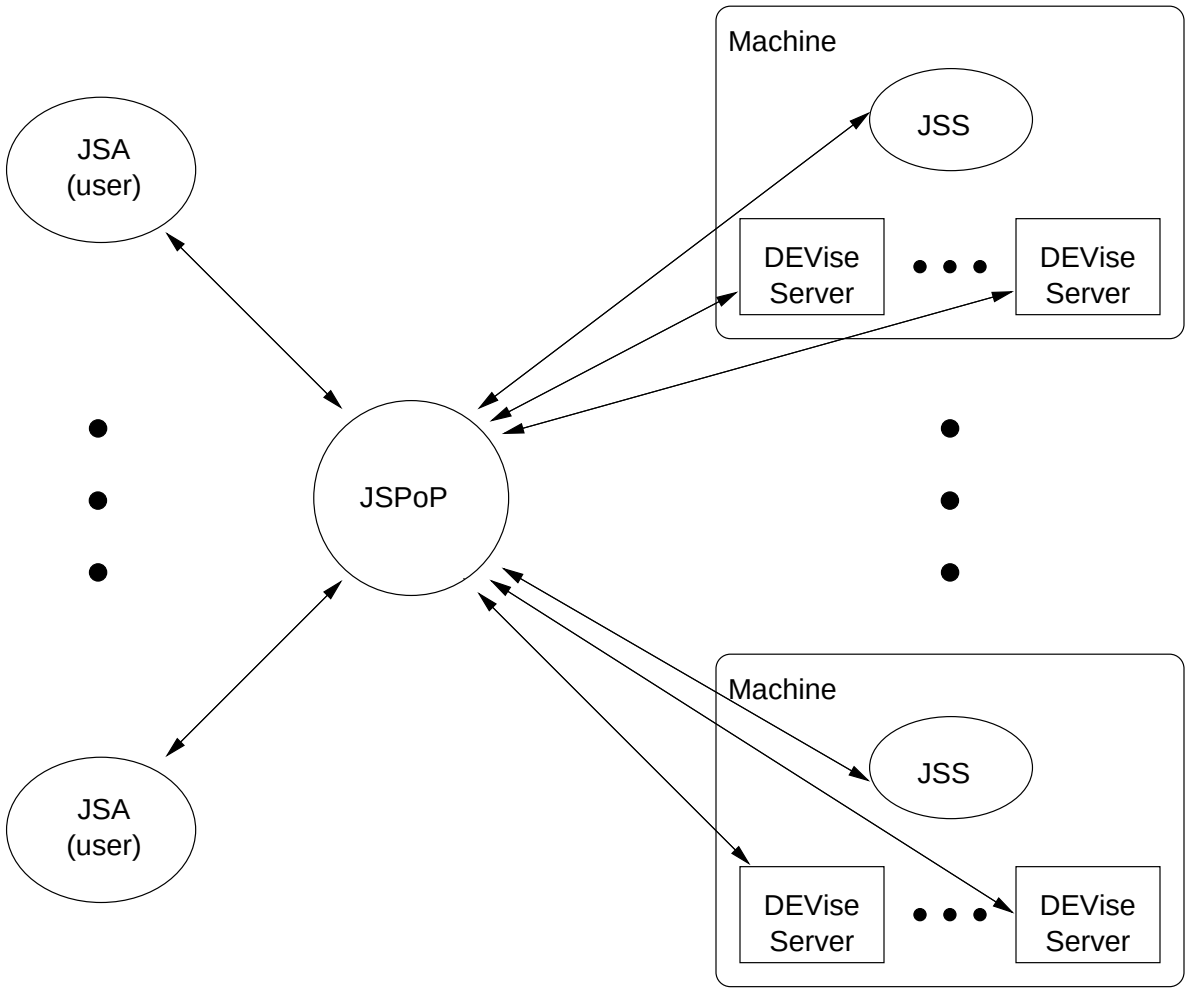

FIGURE 7. JavaScreen communication.

The JSA, running on the user's machine, acts as a virtual screen which displays DEVise presentations and reports user actions to the JSPoP. Almost all rendering is done by the DEVise server; the JSA simply displays the resulting images on the virtual screen. By designing JSA to be as lightweight as possible (the jar file is currently 120 kilobytes), we not only reduce the downloading time of JSA and improve its performance, but also avoid duplication of functionality already existing in the DEVise server.

The DEVise server itself does not support multiple users, which is essential in a distributed environment; that functionality is provided by the JSPoP. The JSPoP acts as a multiplexer. It schedules requests from various JSAs using a priority queue and dispatches those requests to the corresponding DEVise server. When there are fewer DEVise servers than JSAs, the JSPoP gives each JSA (and its user) the impression that it has a dedicated DEVise server by connecting the JSA to a server when the JSA requests a service, and breaking the connection when necessary to service another JSA. (The server's state associated with a particular JSA is saved and restored as needed.) The JSPoP also supports checkpointing, which saves the current state of the JSPoP itself and the current state of all active JSAs and DEVise servers. This ensures the minimum damage to the connectivity between JSA and server if the JSPoP crashes.

The function of JSS is relatively simple compared to the JSA and JSPoP. Its sole role in the JavaScreen setup is to start and stop DEVise servers on request and report these events to JSPoP. Although simple, the JSS is essential in maintaining a scalable and stable distributed environment for DEVise, especially when an unforeseen event happens, such as the crash of a DEVise server.

3.1 DEVise modifications for JavaScreen support

There are two fundamental differences between JavaScreen and non-JavaScreen support in the DEVise server:

1. When running with the non-JavaScreen client, the server draws directly to X Window System windows, while in JavaScreen mode the server sends images to the JavaScreen via a socket, and the JavaScreen actually displays them.
2. When running with the non-JavaScreen client the DEVise server gets mouse and cursor events directly from its windows, whereas the JavaScreen client gets the events, and sends them back to the server as commands.

The first difference causes a significant change in how commands are processed for JavaScreen, as opposed to non-JavaScreen support. When the non- JavaScreen client sends a command that results in queries being run, the client doesn't really care when queries finish, because the drawing is being done directly by the server. Therefore, commands can return to the client before the resulting queries are finished. This doesn't work for the JavaScreen, because the server must wait until all queries have finished, and then send images to the client. This means that the server has a set of commands specifically for the JavaScreen, that wait for query completion, even if equivalent commands already existed for the non-JavaScreen client.

The second difference meant that a number of new commands, with no corresponding non-JavaScreen equivalents, were created to handle mouse clicks, key presses, etc., coming from the JavaScreen. In the non-JavaScreen support, the client is not involved at all when the user, for example, clicks a mouse button in a view.

There are other minor differences between JavaScreen and non-JavaScreen support, but they are not relevant to this paper.

4. APPLICATIONS

In this paper we highlight two of the current applications of DEVise and the JavaScreen: Condor data and Soil Science data.

4.1 Condor visualization

Condor is a project at the University of Wisconsin Computer Sciences Department supporting High Throughput Computing by making use of machine cycles that would otherwise be wasted (at night or when a user is at lunch, for example). Users submit jobs to Condor, and the jobs are run as machines become available. We are currently using DEVise and the JavaScreen to help monitor the state of the UW Condor pool. We look at Condor data in two ways: by architecture and by user.

The architecture data shows, for each architecture in the Condor pool (and for the entire pool), how many of the machines are in "normal" use, how many are being used by Condor, and how many are idle, as a function of time. Summary data is shown for all architectures for a week-long period, and the user uses cursors to select a particular architecture and time period to view in greater detail.

The user data shows, for each Condor user and for all users together, the total number of Condor jobs and the number of jobs running, also as a function of time. There are two Condor user sessions. One shows "thumbnails" of the data for each Condor user, and one allows detailed comparisons between any two users (the users are selected by cursors). These two sessions are

automatically regenerated each night, using the latest Condor data. Typically there are about thirty Condor users active during a given week (which is the time range displayed).

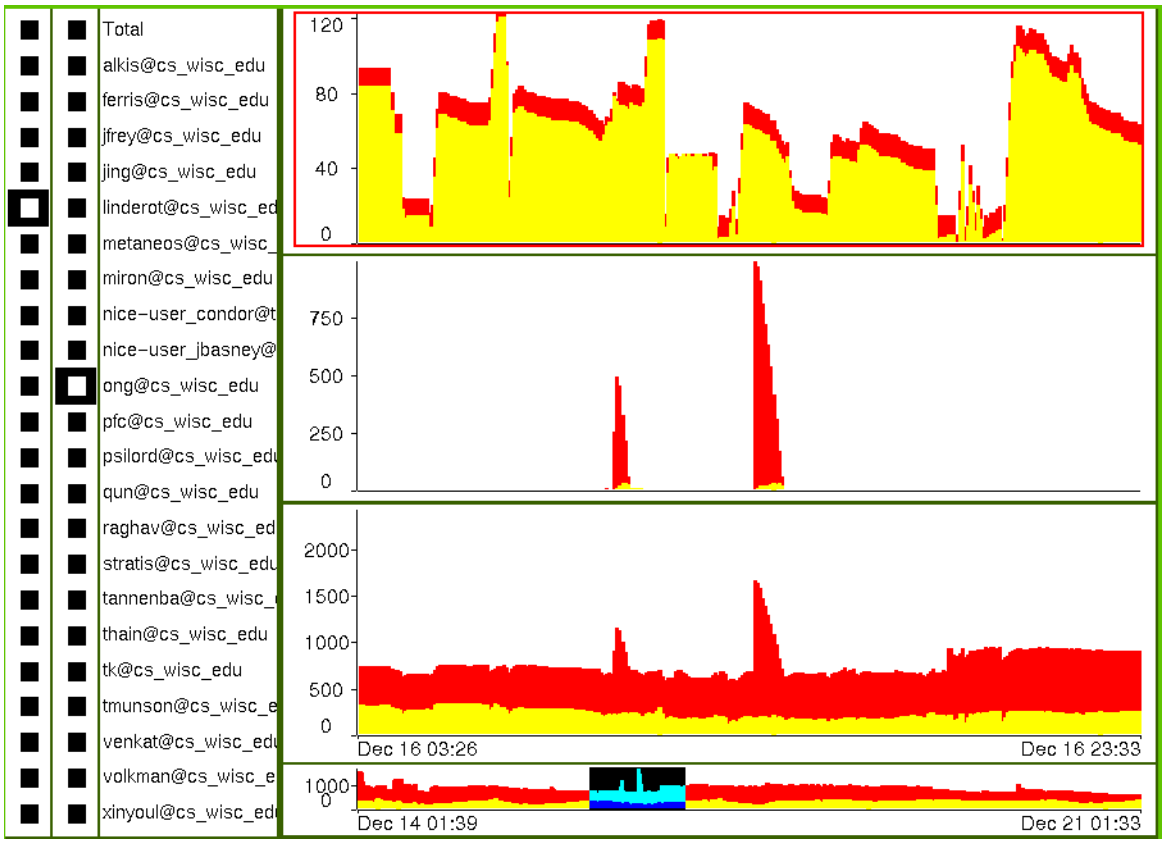

FIGURE 8. Condor user visualization.

The main focus of this session (see Figure 8) is the three large views on the right. The top two views show total jobs and running jobs for the two selected users. The third view shows total jobs and running jobs for all users combined. The time range for these three views is controlled by the cursor in the bottom view. The bottom view again shows the data for all users, but over the entire time period for which data is available. The view to the left of these four views lists all users who ran Condor jobs during the last week. The user can move the cursors in the two left-most views to select which users' data is shown on the right.

4.2 Soil science weather data visualization

We have also worked with the UW Department of Soil Science to make weather data available on the web. The data consists of observations from the Automated Surface Observation Stations (about 104 weather stations in Wisconsin and neighboring states). Data is available for several years, although many individual stations have data for only a portion of that time period. There are seven different values (such as temperature, precipitation, wind speed, and so on) available for each station.

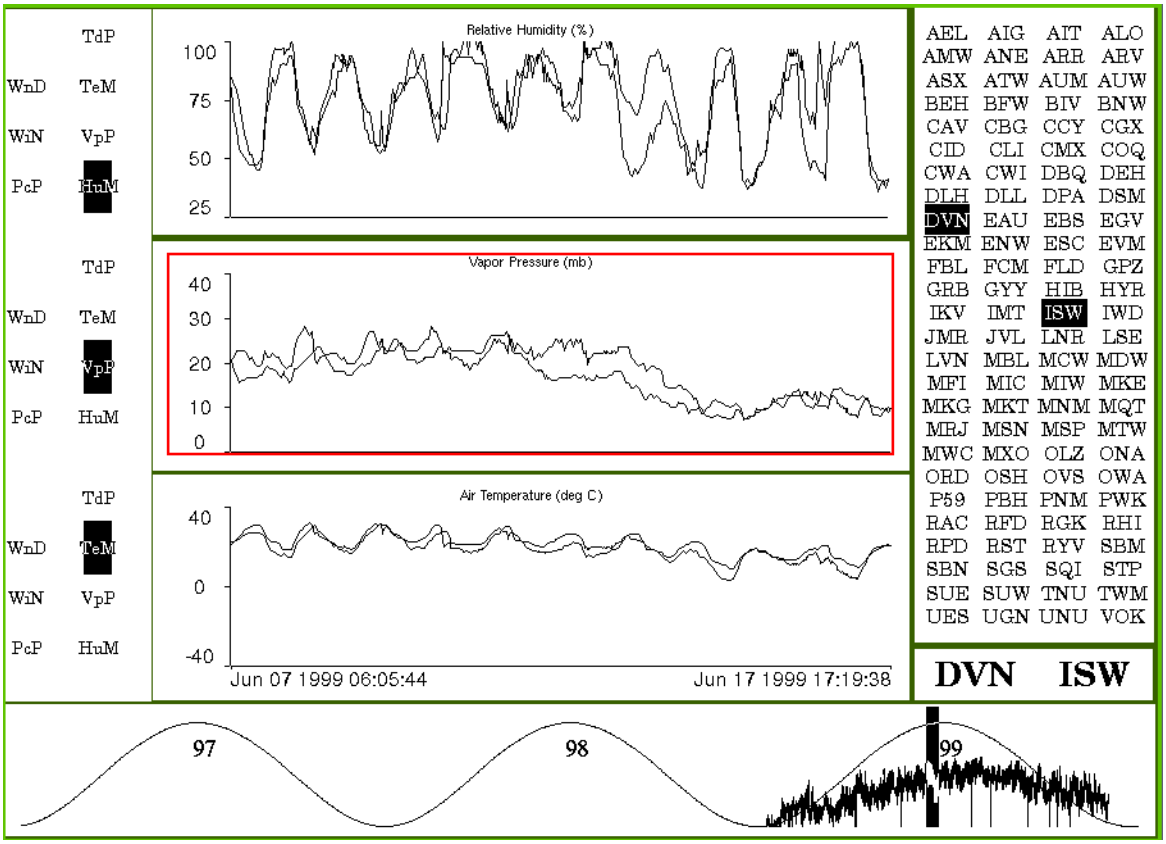

FIGURE 9. Soil science visualization.

The main focus of this session (see Figure 9) is the three large views in the center. These views show three selected weather variables for two selected stations. The cursors in the large view in the upper right are used to select the two stations; the cursors in the three small views on the left select the variables to be shown. The bottom view shows the time range for which data is available for the selected stations, superimposed on the theoretical solar radiation (showing the seasons). The cursor in this view selects the time range displayed in the main views.

5. CONCLUSION

DEVise is a powerful and flexible tool for visualizing and exploring a wide variety of data. By providing a means for easily making dynamic visualizations available via the web, the JavaScreen gives more people the chance to make use of DEVise, both as “data providers” and as “data explorers”. As we make use of DEVise and the JavaScreen for more “real world” applications, we gain valuable feedback that enables us to continue improving both DEVise itself and the JavaScreen.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation awards IIS 9700799 and IIS 9802882.

REFERENCES

1. Alexander Aiken, Jolly Chen, Michael Stonebraker, and Allison Woodruff, “Tioga-2: A Direct Manipulation Database Visualization Environment”, *Proceedings of the 12th International Conference on Data Engineering*, pp. 208-17, New Orleans, LA, February 1996.
2. K. Perlin and D. Fox, “Pad: An alternative approach to the computer interface”, P*roceedings of ACM SIGGRAPH 1993*, pp. 57-64, Anaheim, CA, August 1993.

3. Bederson, B., McAlister, B., “Jazz: An Extensible 2D+ Zooming Graphics Toolkit in Java”, CS-TR-4015, UMIACS-TR-99-24, May 1999.

4. S.F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein, “Interactive Graphic Design Using Automatic Presentation Knowledge”, *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI ‘94)*, pp. 112-117, Boston, MA, April 1994.

5. Steven F. Roth, Peter Lucas, Jeffrey A. Senn, Cristina C. Gomberg, Michael B. Burks, Philip J. Stroffolino, John A. Kolojejchick, and Carolyn Dunmire, “Visage: A User Interface Environment for Exploring Information”, *Proceedings of Information Visualization*, pp. 3-12, IEEE, San Francisco, October 1996.