InfoStill: A Task-Oriented Framework for Analyzing Data Through Information Visualization

Kenneth Cox, Stacie Hibino, Lichan Hong, Audris Mockus, and Graham Wills

Bell Labs/Lucent Technologies
263 Shuman Boulevard
Naperville, IL 60566-7050

{kcc, hibino, lhong, audris, gwills} @ reseach.bell-labs.com

http://www.bell-labs.com/org/11359/spr-vis.html

ABSTRACT

Data analysis involves many tasks, including defining the problem, data retrieval and exploration, and a presentation of results. In this paper, we present *InfoStill*, a new Java-based framework to support the organization and management of data analysis tasks. InfoStill also aids the user in quickly and easily creating presentations of analysis results in the form of linked HTML documents containing data visualization applets.

1. INTRODUCTION

Most data analysis problems involve a number of tasks, including: developing a problem definition; identifying and retrieving appropriate data sources; data cleaning and conditioning; actually exploring and analyzing the data; and eventually presenting the results of the analysis. This process is far from linear; it typically involves many separate threads of investigation, some of which may be discarded as dead ends, together with considerable backtracking.

Previous work in information visualization has traditionally focused on only one or two steps of this process, with particular emphasis on new views or frameworks for data exploration (e.g., [1, 2]). In practice, however, exploration represents only a fraction of a data analyst's time and effort [4]. Analysts need tools to assist with all stages of the analysis process.

In order to address this need, we have designed and developed an exploratory, visual approach to aid users in organizing their data analysis and exploration and in recording their results for later presentation. We refer to this framework as *InfoStill*, short for Information Distillery.

2. INFOSTILL ARCHITECTURE

The InfoStill architecture, shown in Figure 1, consists of the following main software components: the *Data Warehouse*, the *Task Manager*, and the *Presentation Manager*. InfoStill also uses a set of supporting libraries containing tasks, visualization views, and presentations.

The *Data Warehouse* is a repository for the data analyzed by InfoStill. Our current implementation stores the data as SQL tables, accessed using a standard JDBC interface.

The *Task Manager* assists users in organizing their analysis process, breaking down their analysis into a group of tasks. Analysis tasks are stored in the *Task Library*, which also contains more general *task templates* which users can use as starting points in their analysis process.

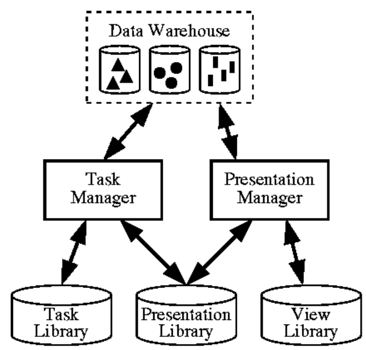

Figure 1. The InfoStill Architecture.

The *Presentation Manager* displays the results of an analysis in the form of presentations. A presentation is both a workspace where users can interactively explore data in a linked views paradigm (similar to EDV [6]), as well as a means to present final results in a reporting paradigm (similar to Crystal Reports [5]).

The remainder of this paper discusses the Task and Presentation Managers in more detail.

3. TASK MANAGER

The current Task Manager prototype includes two main components: an interactive *To-Do* task tree for hierarchically structuring analysis tasks within a standard folder-based hierarchy, and a *Task Properties* dialog box for specifying and editing information such as the goals, data, presentations, and notes associated with a given task.

3.1 To-Do Task Tree Window

The To-Do task tree window, shown in Figure 2, supports users in organizing their analysis in several different ways, such as by importance or along designated lines of inquiry. The window is composed of a set of *menus* and a *main task tree area*.

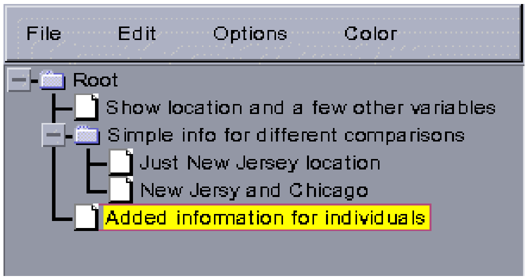

Figure 2. To-Do task list window.

To-Do Task Tree Menus. Users can create, open, and save task trees through the *File* menu. (These task trees are saved and retrieved from the Task Library, which resides on a SQL database server.) Once users have an open task tree, they can add, insert, and delete items and folders from the task tree through the *Edit* menu. The *Options* menu provides simple viewing and selection options for the task tree. The *Color* menu allows users to set the color of any node, allowing users to indicate information such as the priority of tasks or the importance of task results.

Main Task Tree Area. Task items are hierarchically displayed in the main task tree area. Through direct-manipulation and drag-and-drop interfaces, users can edit the names of items and folders as well as re-order items according to their preference.

3.2 Task Properties Dialog

Users can edit and update task properties by double-clicking on an item in the To-Do task tree. This will bring up the Task Properties dialog box (see Figure 3) for the selected task. In the top portion of the dialog box, users can specify basic information about the task, including a descriptive *name*, its primary *goal or hypothesis*, and a basic *strategy* for achieving that goal.

The central portion of the Task Properties dialog box is used for creating exploratory presentations. Here, we use the term *presentation* to refer to an interactive exploratory web document including Java applets of linked views, similar to LiveDocs [3]. Presentations can be used for both data analysis and the final presentation of results. A task can have several presentations associated with it, where the presentations are hierarchically organized and linked as indicated in the *Presentations* area of the dialog box.

Users edit the presentation using direct manipulation. New nodes (representing sub-folders, notes, user inputs, and views) can be added by clicking on the icons above the tree, or by dragging from those icons into the tree. Drag-and-drop can also be used to move and delete nodes.

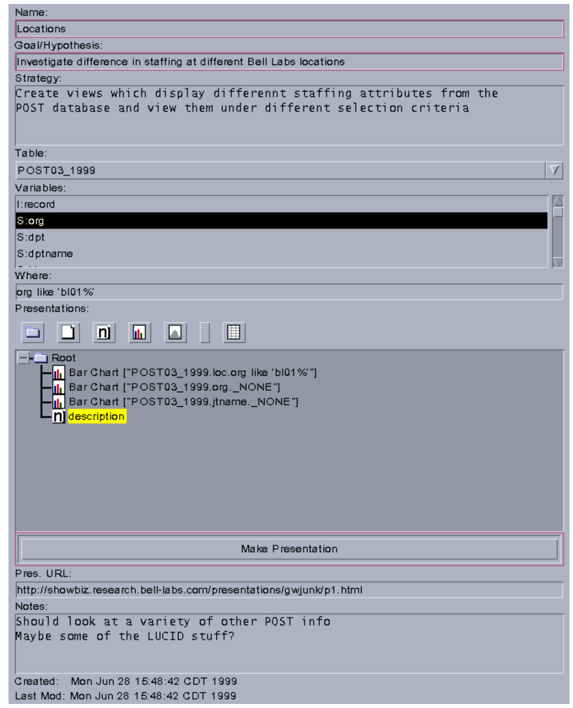

Figure 3. Task properties dialog box.

Double-clicking on a notes node brings up a dialog box to edit the text of the note. Double-clicking on a view node brings up a dialog for editing the view properties, such as the data that the view displays. These properties can also be set by direct manipulation; for example, the user can assign data to a view by selecting a database table, field, and where-clause from the lists above the tree and dragging into the view node icon.

Once all or part of a presentation tree has been specified, users can generate HTML for the presentation by clicking on the *Make Presentation* button. The root of the presentation tree is saved at the URL shown in the *Pres. URL* field.

Finally, the bottom portion of the task properties dialog box provides a space for users to take *notes* about the given task and indicates the dates in which the current task was *created* and *last mod*ified.

Overall, the Task Manager relieves users from the burden of having to organize their analysis tasks by hand, thereby allowing them to spend more time focusing on their actual analysis goals.

4. PRESENTATION MANAGER

4.1 Overview of the Presentation Manager

In our prototype, we use a Java-capable HTML browser to display presentations. This has obvious advantages for the distribution and accessibility of presentations. In this implementation, the presentation manager consists of a set of Java libraries used by applets embedded in the HTML documents displayed by the browser.

Each folder in a presentation tree built using the Task Manager is translated into one HTML page. The contents of the folder are translated into appropriate HTML elements:

- *Notes* and annotations become ordinary text elements.
- *Views and inputs* become applet elements, with associated header markups to label the elements.
- *Sub-folders* become links to the HTML pages represented by the sub-folder. Each page also has a link to its parent and to the page at the root of the presentation tree.

Each page also has an additional hidden applet element, the *loader*. The loader applet defines the shared objects used on the page and, for shared objects obtained from the external database, describes how these objects are to be created. A shared object can be one of three types:

- *data*, or columns from the database (i.e., selected fields from a data table);
- *links*, or objects which are used to implement case-based view linking; and
- *inputs*, or objects which represent input choices or strings entered by the viewer of the presentation.

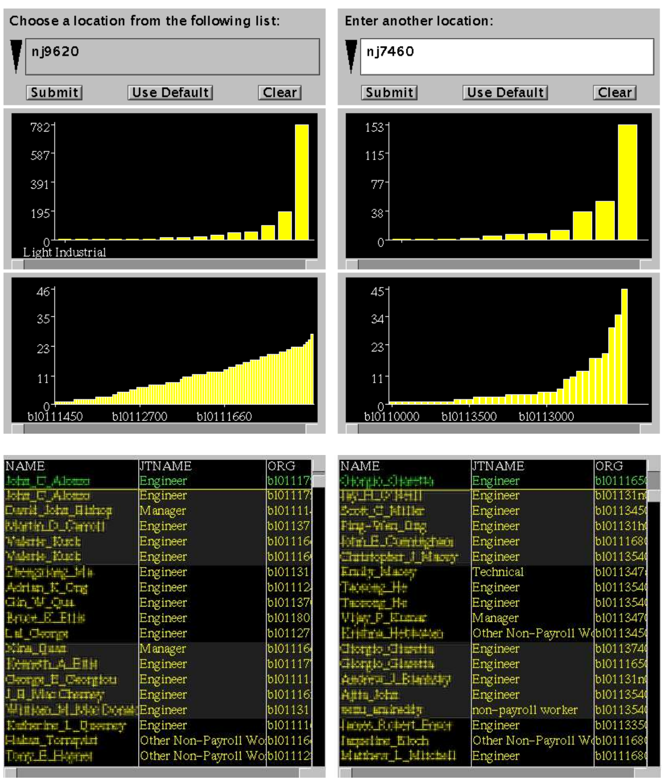

Figure 4. A sample presentation. The names in the lists have been deliberately rendered illegible.

Figure 4 shows a browser view of a sample page which uses all three types of applet elements—the loader, the input, and the view. (We omit the headers and other textual elements, generated from the Task Manager presentation tree, that are also on the page.) In the remainder of this section, we use this figure to illustrate the functionality of the presentation manager runtime package.

The loader applet manipulates *loads*, which control two key aspects of the shared objects associated with a presentation:

- what data is loaded from the data warehouse, and how it is to be loaded (e.g., whether it is to be aggregated);
- the mapping of shared objects to the logical names by which they are known to the applets.

This enhances both the flexibility and efficiency of the presentation in that the loader can support multiple loads within a presentation web page as well as sharing loads across pages (i.e., data will not be reloaded from the warehouse if it has been previously loaded). Figure 4, for example, uses two loads—one for the left views and one for the right views. This allows users to easily compare the same views of different data.

4.2 Example Use of Loads and Views

In this section, we concentrate on the left set of views of Figure 4. These views use a single load, which is defined in the loader applet's HTML parameters. This load brings in three columns of data from an employee database: the employee's name (NAME), job title (JTNAME), and organization code (ORG). Each of these columns becomes a shared data object, and in addition a shared link object is created by the load and associated with the three columns.

The three view applets access the shared objects by name (as specified in the applets' HTML parameters) and display the data; the top bar chart displays the job title, the second bar chart the organization code, and the values list at the bottom shows all three columns. These views are linked together by the shared link object, so selection actions in one view are propagated the other views.

In this example the load is further controlled by the input widget at the top of the left column. The widget allows the user to select from a drop-down list of employee location codes (LOC). This selection modifies a shared input object (also defined in the loader HTML) which is used in the where-clause of the SQL query that reads the data. Thus, the user can dynamically change the data loaded from the database using the input widget.

When a user chooses a different location code from the input widget, the loader checks its cache for the requested data and only retrieves the data if it is not already available. The loader performs data retrieval by translating the data request into a SQL query; in the example shown in the figure, the query is

```
SELECT NAME,JTNAME,ORG FROM
  POST03_1999 WHERE LOC='nj9620'
```

This translation to SQL is of course done transparently so that users who view presentations are not required to enter SQL commands in order to analyze and examine different subsets of data. Once the loader has retrieved the newly requested data, it broadcasts the update to the appropriate views and the views are then redrawn with the new data.

4.3 Using Multiple Data Loads to Compare Data Subsets

The right column of Figure 4 is defined almost identically to the left-hand side, with two differences: 1) each object has a different logical name, and 2) the input object is defined to allow text entry so that the user can input any location desired.

In the resulting interactive presentation, readers can quickly and easily compare two locations with respect to the organizations represented at each location, the different types of jobs held and even to extract names, phone numbers and email addresses if needed. In the sample comparison presented in Figure 4, the shape of the upper bar charts show that job patterns are very similar. The lower bar charts show that although the location on the left has a fairly uniform spread of department sizes, the location on the right has many more smaller departments and a few very large ones.

Inputs can support other types of comparisons, since input values may appear anywhere in the SQL query, not just in where-clauses. For example, `POST03_1999` is only one snapshot of the employee database (for March, 1999). We could provide a shared input which would be used for the table name, and thus allow the user to compare different snapshots and see how the data changes through time.

HTML presentation pages can be generated from the presentation tree of the Task Manager very rapidly. This allows the user of the Task Manager to keep an open browser pointing to the generated HTML, and re-load the pages after changes are made. The user thus sees the exact same presentation materials that others will see when they browse the final presentation, and can explore the data set in the same way as those later users can.

Overall, the presentation manager is capable of supporting a potentially complex combination of views, loads, and inputs. The end-user is shielded from being overwhelmed by this complexity, however, since 1) the HTML specification of a presentation is automatically generated through the Task Manager and 2) underlying SQL queries are automatically handled by the data loader.

5. SUMMARY AND FUTURE WORK

The InfoStill approach is designed to support users in several key data analysis activities, including:

- keeping track of their lines of inquiry, including information such as their goals and hypotheses, current strategies, and related notes;
- quickly and easily composing annotated interactive, hierarchical presentations for exploring data;
- sharing and presenting results of their data analyses in a standard Web format.

We are applying our prototype implementation to real data analysis problems with the twin goals of improving the prototype and furthering our understanding of task-oriented data analysis.

ACKNOWLEDGMENTS

We wish to thank the members of the Collaboratory Project, including Jim Herbsleb and Beki Grinter, as both sources of interesting ideas and as testers of our prototypes.

REFERENCES

[1] Ahlberg, C., & Shneiderman, B. (1994). Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. *CHI'94 Conference Proceedings*. NY: ACM Press, 313-317.

[2] Carlis, J.V. and Konstan, J.A. (1998). Interactive Visualization of Serial Periodic Data. UIST'98 Conference Proceedings. NY: ACM Press, 29-38.

[3] Eick, S., Mockus, A., Graves, T., Karr, A. (1998), A Web Laboratory for Software Data Analysis. In *World Wide Web*, v. 1 n. 2, pp. 55-60.

[4] Hibino, S. (1999). Task Analysis for Information Visualization. In *Visual Information and Information Systems: Third International Conference, VISUAL'99* (Huijsmans, D.P. and A.W.M. Smeulders, Eds.). Berlin: Springer-Verlag, 139-146.

[5] Seagate. Seagate Crystal Reports. 1999, http://www.crystalinc.com/crystalreports/

[6] Wills, G.J. (1995). Visual Exploration of Large Structured Datasets. In *New Techniques and Trends in Statistics*. IOS Press, 237-246.