Collecting and Harnessing Rich Session Histories

Howard Goodell, Chih-Hung Chiang, Curran Kelleher, Alex Baumann, Georges Grinstein
University of Massachusetts at Lowell
hgoodell, cchiang, ckellehe, abaumann, grinstein@cs.uml.edu

Abstract

To be most useful, evaluation requires detailed observation and effective analysis of a full spectrum of system use. We have developed an approach and architecture for in-depth data collection and analysis of all use of a visualization system. User interface components in a large visualization and analysis platform automatically record user actions, and can restore previous system states on demand. Audio and text annotations are collected and indexed to states, allowing users to find a comment and restore the system state in which they made it; then explore actions before and after. History is visible as data; so a variety of visual displays and analysis techniques may be used to develop insights about the user's experience. States of any part of the interface may be analyzed separately. Actions are categorized in a taxonomy as the user interface is built, allowing comparison of similar patterns in all tools. History data can co-exist with other data during data exploration, supporting further individual or group data exploration.

Keywords--- User monitoring, session history visualization, session history, voice annotation

1. Introduction

At the most basic level, evaluation requires observing use of a system, analyzing our observations, and correlating results with some standards for performance or goals for improvement. The more broadly and accurately we can observe, and the more flexibly we can represent and model and analyze the data, the more reliable our conclusions can be.

Manual experiments and observations can achieve excellent accuracy, but at a cost that limits them to typical users performing representative tasks. Automatic recording such as web server logging can physically record all use of a system, but their lack of precision and standardization often makes it difficult to understand user behavior in detail.

Our research group has developed an approach for recording in-depth session histories, and applying them in many ways to benefit users and facilitate analysis. An architecture realizing this approach has been implemented in the core framework of a data visualization and analysis platform, the Universal Visualization Platform (UVP) [3]. The UVP is designed to make it easy to create new visualization and analysis tools, called *tools*. Because of the session history features provided by the framework, these tools automatically record and interact with session histories. Our approach is not limited to visualization systems, although visual displays are helpful to represent and interact with recorded history.

2. Visual history exploration

Our session history infrastructure provides automatic recording, restoring, annotation, and flexible visualization and analysis of user actions. These capabilities provide a basis for some novel system evaluation techniques.

Our system's primary theoretical basis is an extension of the Generalized Data Exploration (GDE) model of J. P. Lee and Grinstein [8] and J. P. Lee, [7]. User actions are recorded as changes to system state. Each action is categorized in a visual data exploration taxonomy as the system is built; so state changes anywhere in the system can be compared in a principal way [4]. Metadata including audio and text annotations is associated with the states in which it was recorded. Users or researchers reviewing a session history may also add annotations to any state.

Users can do a number of things with the histories of their own and others' sessions. The two major ones are immediately restoring any prior state of their current session (often called time travel) and replaying histories of previous sessions.

Users and researchers may also view and analyze session histories as data. History datasets are views of the saved session states with a row (record) for each user action and a column (dimension) for each persistent value that changes. Users and researchers may analyze these datasets with all the tools in their discovery system to locate interesting states. Any state located in the history dataset may also be immediately restored, and

Proceedings of the Information Visualization (IV'06)
0-7695-2602-0/06 $20.00 © 2006 IEEE

states before and after it explored. They may take new actions from restored states, both to evaluate the choices made previously or to continue the exploration of the dataset. By restoring records of past exploration sessions on a particular dataset, users can evaluate the ways they and others have explored it in the past, and extend those explorations from any point.

We provide UVP built–in tools to analyze the history of part or all of the system. For example, we detect actions that return to previously-explored states and compute metrics on the exploration path such as the GDE exploration depth (number of steps without return to prior states) and width (the number of paths explored to the current depth) and rates of forward progress (Section 5 below).

Audio and text annotations are correlated with actions; so evaluation protocols like "thinking aloud" can be analyzed in the context of the user actions and system states that accompanied the words [1]. Users or researchers viewing a session may go back and forth freely between browsing and searching annotation text, viewing patterns in the history visualized as data, and restoring and navigating in the history.

3. Interacting with history

The system provides three main functions for interacting with history: monitoring, restoring, and replaying previous sessions. Audio/text annotation is also linked to history bidirectionally (read/listen to annotation for history state; or search annotation and restore state).

3.1 State-based monitoring

3.1.1 Persistent data values. The system provides tool writers a set of user interface components that are instrumented to record changes and restore themselves to previous states on demand. For example the UVP provides a set of GUI widgets we call "UVPWidgets", such as `UVPRadioButton` and `UVPComboBox`. The state of these user interface components is maintained in "UVPValues": custom persistent versions of Java data objects like `UVPFloat`, `UVPString`, and `UVPColor`. The custom GUI objects and the UVPValues that hold their state implement the Observer-Observable pattern [1]. The UVPValues are Observable, and the GUI objects whose state they maintain are Observers notified when the UVPValue's state is changed.

A different design pattern describing this process is the Model-View-Controller (MVC) pattern [6]. In MVC terms, the UVPValues are the Model and the UVPWidgets combine View and Controller functions. When the UVPValues (Model) are restored to their value at a previous time, they notify the GUI widget to change its appearance (View) and take other actions (Controller) such as repainting the display that this value change would produce if the user had changed the value.

For example, consider a list box that maps data dimensions to the X or Y axis of a scatter plot display. The list box is an instrumented UVPWidget that maintains its selected value (a string or integer) in a UVPValue, that has the list box widget as an Observer for its value changes. Note that tool designers simply use a UVPWidget that encapsulates these interactions, and in most cases they do not need to be aware of them. Each time a new list element is selected, the UVPValue adds the change to its list of previous values, stamped with the current time. When the next inferred user action is recorded and logged (see next section), this UVPValue will be included in its list of changed values.

3.1.2 Inferring user actions. Most user actions only change a single persistent value; so there is a 1:1 correspondence between actions and state changes. However some actions cause multiple value changes, such as an outer window resizing that causes contained-window resize events, or the opening of a new visualization tool that causes a number of individual UVPWidgets and values to be initialized. We use several procedures and heuristics to avoid treating as real these system events which do not represent genuine user actions but rather are software artifacts of the propagation of a single user action.

Our primary heuristic uses human response time. Since it is difficult for human beings to react individually to events occurring less than about 50 milliseconds apart, we assume that events that occur faster than this are system-generated. Events that occur more slowly are considered to each be a genuine user action. The result of this implementation is that each inferred UI event object that is recorded contains all the state value changes that resulted from the event. Although this heuristic is not perfect, it performs very well. Figure 1 shows the process, culminating in writing the event to an XML log.

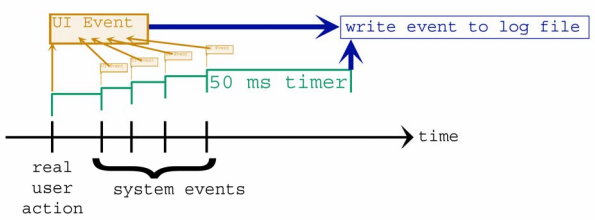

Figure 1: Time Heuristic to Infer User Actions

3.2 Restoring previous states

We use Rekimoto's term *time travel* for return to previous states, often referred to as *undo and redo* [9]. Unlike systems that record actions, our time travel restores states rather than repeating or undoing actions. Users are not limited to undoing or redoing a step at a time; they can time travel directly to any point. In MVC language, the system restores the Model of each component; then updates the View to represent it.

The system's Models form a hierarchy, from one Model that contains the state of the entire system, to models for major system components such as visualization tools, down to a Model for a single GUI component. We use the Visitor pattern to recursively restore ([2], p. 331). After each Model is restored to the selected state, the View of its GUI elements updates.

Users can select past states to restore using special-purpose tools like history list (Figure 2) and history tree visualizations (see Section 4). They can visualize history as data and use any visualization to locate a remembered or other interesting point in the history. Then they can "time travel" (restore) the system to that state. In Figure 2, note that the history box's "player" metaphor includes not only single steps forward and back, but long steps, which go to the next state explicitly "bookmarked" for reference or that has an active annotation (Section 0). It also allows direct "time travel" to any past state.

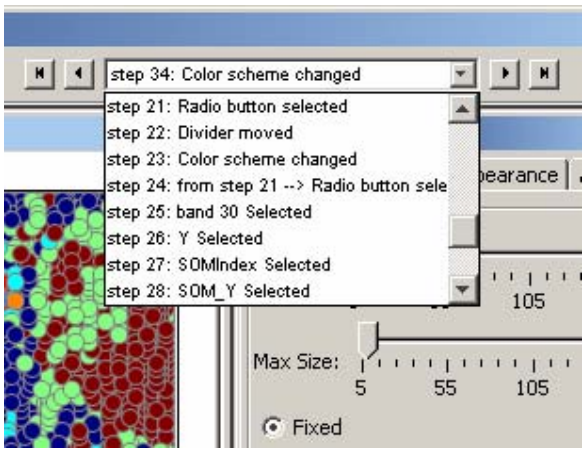

Figure 2: History list *time travel* to past state

3.3 Replaying actions

A log file produced in one session can be "replayed" into a later one. The log file contains a unique ID for each changed UVPValue, containing its tool instance and name. For example, the current Y axis selection for a session's first instance of the ScatterPlot tool would have the unique ID "`ScatterPlot_1.colYUVP`". In replay, the unique IDs from the previous session are translated: so if there are already two ScatterPlot instances, references to this identifier become "`ScatterPlot_3.colYUVP`". As the correct instance of each UVPValue is restored to the value recorded in the log file, the GUI widget that is its Observer restores the corresponding View elements. A log file may be output as an XML file.

Replayed actions become part of the later session, but with the timestamp of the previous one. Users can time travel into the past session and take new actions, which become part of the current session. This is done consistently by using references to step numbers rather than clock times, and translating historical steps to the numbering of the current session, just as is done for tool instances. For example, say that a previous session was replayed starting at Step 53 of the current session. If one of the replayed steps is, "Return to step 11", it is translated into, "Return to Step 64", and a restore operation will go to the correct step. If the current session is replayed in some future session, "Return to Step 64" will be further modified by adding the starting event# of the log replay in *that* session; so the original "Return to Step 11" will still execute correctly.

Figure 3 summarizes the actions of recording, restore, and replay. Note that all three actions are based on the persistent data elements, "UVPValues", which maintain and restore past states.

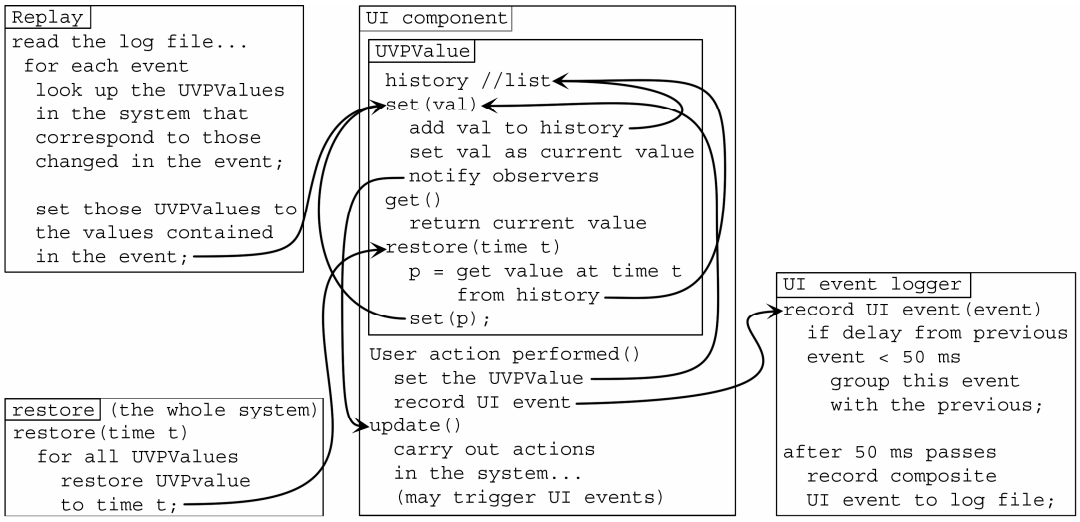

Figure 3: A summary of recording, restore, and replay

3.4 Voice Annotations

Integrating voice annotation with automatic monitoring provides a natural and unobtrusive means of gathering information from users. This information complements the precise but inarticulate records of their actions. As users express their thoughts and findings during the discovery process, the ideas they express form contextual links between their abstract thoughts and the concrete program states in which they expressed them. Using text-to-speech software, automatic monitoring of user actions can be supplemented by automatically-recorded annotations, creating a more balanced record.

Our initial implementation treated each annotation as an action that became part of the session history. Although this approach provided retrievable audio and text located at the correct places in the history, it did not allow other actions to take place during annotation. We now record annotation concurrently with other actions, with time tags linking to actions. This is makes the annotation process more natural, enabling users to annotate any part of the session without a special effort.

Users' annotations may relate directly to their current actions, but they may also refer to unrelated aspects of the discovery process, the performance of the software, or any other topic. Therefore flexibility and interaction is required to understand them. Since annotations are provided as both audio and text, several reading modes are possible. In one, the user's audio recording is replayed while program actions replay in real time. In another, the converted or user-entered text is searched and the contemporaneous actions analyzed manually or automatically. The combination provides richer information than either record alone. For instance annotation text may be searched for a keyword, and the state(s) in which the word was said can be compared. This procedure allows searching for abstract ideas or remembered terms, and the concrete states that related to those ideas or triggered the utterances can be observed.

Analysis of annotation text and the program states together may provide many clues for improving an interface. For example, if certain keywords that are reliably used in annotating particular operations, these operations might usefully be automated by speech commands. Obviously any annotations about the performance of the software, such as users' emotional responses to bugs and missing features, can be used (by the programmers or machine learning algorithms) to improve system performance and track bugs.

Several issues must be considered for this technique to be successful. First, if the annotation process runs continuously, it would be very helpful to have heuristics to distinguish what users intend to be voice annotations from their speaking to colleagues in the room or by telephone. Second, continuous recording raises privacy issues. Are users comfortable with whatever they say becoming text and recorded audio within the application? The principle we use in designing all session recording is to put users in charge: they may turn off recording or delete audio files, and delete or edit the converted text.

Applying our tool to evaluation has interesting potentials. "Thinking aloud" protocols can be contemporaneous or retrospective, with users either speaking as they work or offering explanations as they view a videotape of their actions [5]. Our infrastructure supports either mode, with additional possibilities. Users can interrupt action replay to try other possibilities and browse back and forth as they add to their previous remarks. Then can also visualize their history and explain it from an overview perspective (next section.)

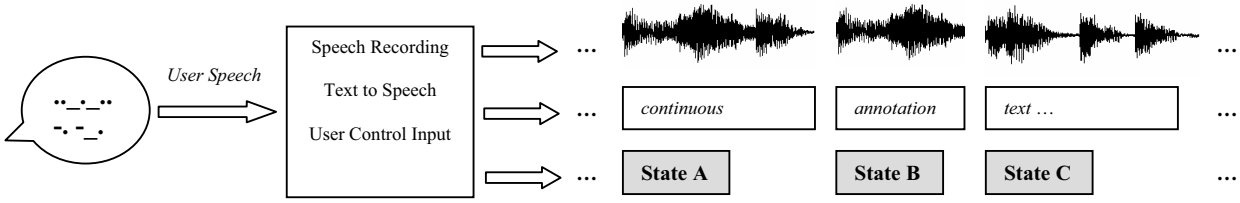

Figure 4: Recorded audio and text annotations are linked to states of session history.

4. Visualizing history data

In the main UVP example research application Vizit, a Session Graph tool appears automatically at the bottom of the screen when a dataset is opened (Figure 5). This graph has a vertex for each state and an edge for each action. Users choose the data labeling each state. Much more data is available by probing individual vertices. Unlike the History List, the Session Graph visualization does not automatically add a state vertex each time users take an action. It detects when they return to a prior state – taking actions and then reversing them – and creates a branching tree structure from left (the start of the session) to right (the "deepest" operation, created by the longest string of actions that explored new states rather than returning to old ones.) Operations that return to previous states do not create an edge, but simply re-select the previous state. Probing a Session Graph vertex reveals all its entries and exits along with metrics (Section 5) and other details. The operator may also time travel to any viewed state.

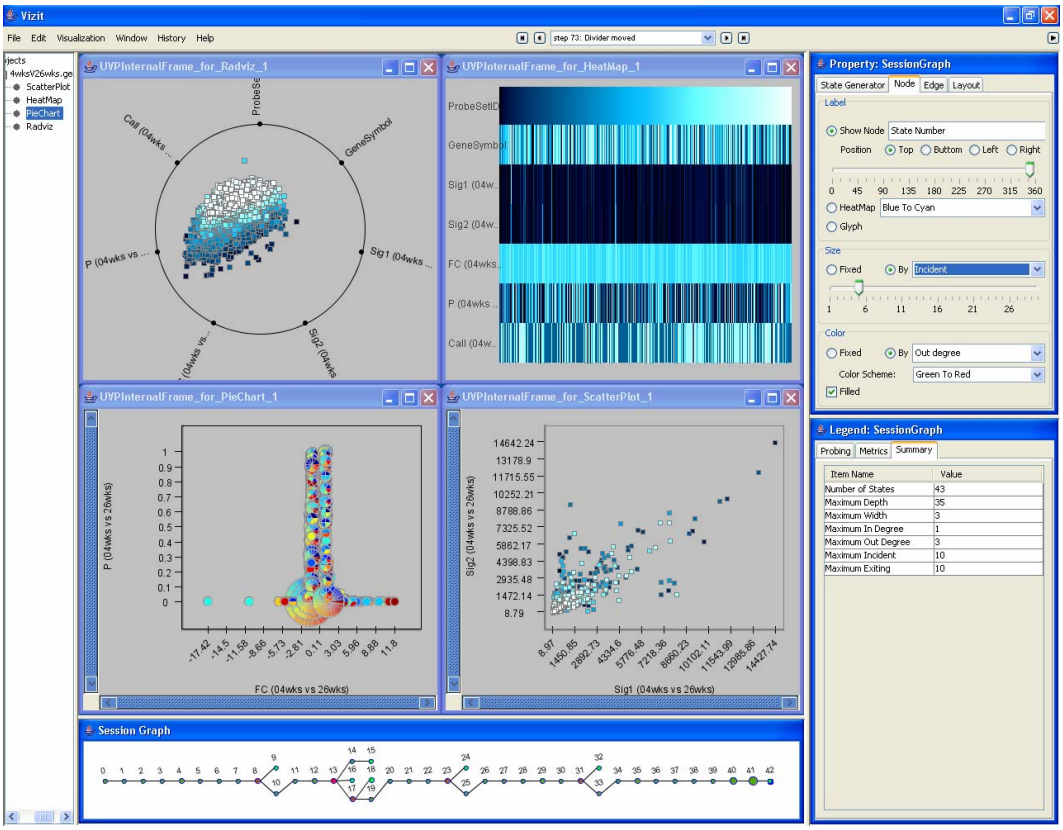

Figure 5: Visualization session with session graph representation. Note branching in session graph.

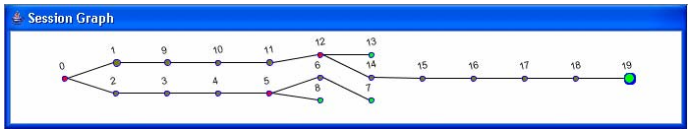

Figure 6: Pie Chart states (only) in above session. Note the much simpler graph structure.

The default Session Graph or other visualization is of the state of the entire system. However it is often useful to display the states of just part of the system: a tool, a selection of the operation taxonomy, or even an individual control. Figure 6 shows the states of one tool, PieChart, within the above multi-tool session.

In our architecture, a Model object at any level of the hierarchy can return a view of itself as a dataset. Because these views do not actually copy the history data, it is feasible to view a flexible variety of Model levels. This flexibility is helpful to support users, who may remember the tool or control they used for an action hours ago, and can understand its much simpler history more easily than the full system state graph. It also opens interesting avenues for evaluation (see below).

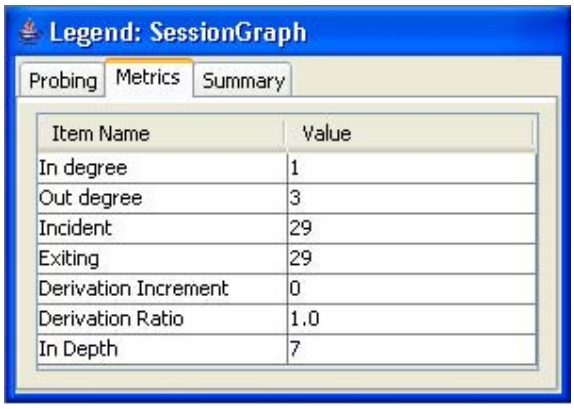

Figure 7: Metrics on the Selected Vertex

5. Using the GDE metrics for evaluation

The GDE defines two kinds of metrics on the session graph: vertex and path. Figure 7 shows metrics for the selected (probed) vertex in the Session Graph.

Vertex metrics are computed on a single node of the graph, based on in- and out-degree and total visits. They may be used to classify vertices, for example “landmark vertices” the user repeatedly returns to.

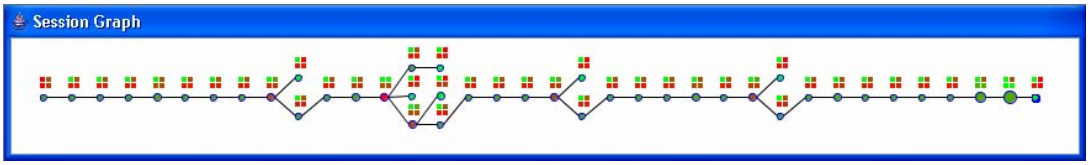

Figure 8: Heatmaps show vertex metrics: upper LH in-, RH out-degree; lower LH incident, RH exiting.

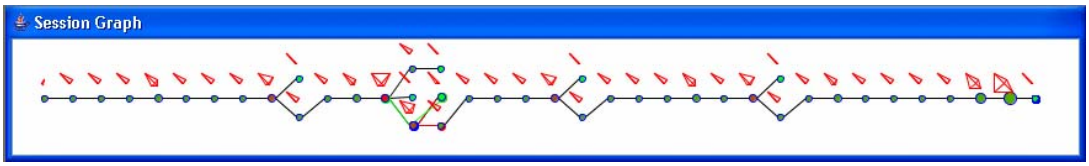

Figure 9: Star glyphs show vertex metrics: upper LH in-, RH out-degree; lower LH incident, RH exiting.

Figure 8 and Figure 9 represent the metrics for each vertex on the state graph with tiny glyphs or icons. Figure 8 uses a heatmap and Figure 9 a Star Glyph.

Path metrics, computed upon all or a connected subpath of derivations, include path depth (number of transitions from the source to the node in question) and breadth (number of different states visited in parallel at the same depth from the source). Some of these are visible in the lower right-hand corner of Figure 5. Many other vertex and path metrics such as rates of progress towards results are based on these simple foundations.

Several of the GDE metrics are directly usable for evaluation. For example the depth and breadth metrics, and time-based metrics such as exploration rate, indicate whether the analyst is able to make rapid progress with this data and tool set.

The metrics for each Model may be viewed and analyzed as a dataset of unique states, similar to the view of its history as a dataset. This general approach provides interesting flexibility for evaluation. If one tool or widget shows much broader paths (indicative of users trying many different approaches that probably didn't work) or slow exploration rates, this may indicate usability problems with that part of the program. The metric dataset for all or selected components can be analyzed visually or statistically to quickly locate targets for closer scrutiny. Evaluators may view the state history for questionable components and time travel to states where users appeared to have trouble, looking at their annotations, seeing the state of the interface they saw, and stepping forward and back to find where they went wrong.

Conclusions and future work

We have provided a model of data exploration environments that monitor and interact with session history in flexible ways. We've described its implementation and highlighted some of the issues. We described the advantages of our approach by showing the support for viewing and calculating metrics on the history of the whole system or individual parts, for interactions with the session history data, and for voice annotation and search. Replaying multiple sessions for analysis and moving freely between analyzing the histories as data and time traveling into viewed states to browse or continue other users' explorations provides a rich environment for exploration, for system evaluation, and for user support.

Our history tools provide a basis for many other uses of session history. We are currently exploring modeling and statistical analysis of multiple sessions and user groups.

Acknowledgements

The authors thank J.P. Lee for his helpful discussions, and UVP team members who contributed ideas and implementations to the session history features, especially Alex Gee, Chris Lawrence, and Jon Victorine.

References

[1] Boren, M.T. and Ramey, J. Thinking Aloud: Reconciling Theory and Practice. *IEEE Transactions on Professional Communication*, *43* (3). 261-278.

[2] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns*. Addison-Wesley, 1995.

[3] Gee, A.G., Li, H., Yu, M., Smrtic, M.B., Cvek, U., Goodell, H., Gupta, V., Lawrence, C., Zhou, J., Chiang, C.-H. and Grinstein, G.G., Universal Visualization Platform. in *Visualization and Data Analysis 2005*, (San Jose, California, USA, 2005), SPIE, 274-283.

[4] Goodell, H., Chiang, C.-H. and Grinstein, G.G. Taxonomized Automated Monitoring of Interactive Visualization, Computer Science Department, University of Massachusetts at Lowell, Lowell, Massachusetts, 2006.

[5] HAAK, M.J.V.D., JONG, M.D.T.D. and SCHELLENS, P.J. Retrospective vs. concurrent think-aloud protocols: testing the usability of an online library catalogue. *BEHAVIOUR & INFORMATION TECHNOLOGY*, *22* (5). 339-351.

[6] Krasner, G.E. and Pope, S.T. A cookbook for using the model-view controller user interface paradigm in

Smalltalk-80. *J. Object Oriented Programming*, *1* (3). 26-49.

[7] Lee, J.P. A systems and process model for data exploration *Computer Science*, University of Massachusetts at Lowell, Lowell, Massachusetts, USA, 1998, 245.

[8] Lee, J.P. and Grinstein, G.G. An Architecture for Retaining and Analyzing Visual Explorations of Databases. *Proceedings of IEEE Visualization 1995*. 101-109.

[9] Rekimoto, J., Time-Machine Computing: A Time-Centric Approach for the Information Environment. in *ACM Symposium on User Interface Software and Technology*, (1999), 45-54.