2001

Enhancing Data with a Branching HIstory of User Operations

Mark Derthick
Carnegie Mellon University

Steven F. Roth
Carnegie Mellon University

Knowledge Based Systems, 14(1-2):65-74, March 2001.

Title: Enhancing Data Exploration with a Branching History of User Operations

Authors: Mark Derthick and Steven F. Roth

Address:

Mark Derthick

Human Computer Interaction Institute

Carnegie Mellon University

Pittsburgh, PA 15213

(412) 268-8812 (telephone)

(412) 268-1266 (fax)

mad@cs.cmu.edu

Steven F. Roth

Robotics Institute and Human Computer Interaction Institute

Carnegie Mellon University

Abstract

Backtracking and investigating alternative scenarios are a integral parts of exploratory data analysis. Yet today's interfaces cannot represent alternative exploration paths as a branching history, forcing the user to recognize conceptual branch points in a linear history. Further, the interface can only show information from one state at a time, forcing users to rely on memory to compare scenarios. Our system includes a tree-structured visualization for navigating across time and scenarios. The visualization also allows browsing the history and selectively undoing/redoing events within a scenario or across scenarios. It uses the AI formalism of contexts to maintain multiple, possibly mutually inconsistent, knowledge base states. Cross-context formulas can be written for explicit scenario comparison, including visualizations of scenario differences.

Keywords: Undo, Exploratory Data Analysis, Context

1. Introduction

When I stopped using a mechanical typewriter and started using a word processor I marveled at its ability to rub out characters. With today's powerful computers it is possible to record all the actions performed through the user interface for a wide variety of applications. Yet current interfaces limit the usefulness of all this recorded information to little more than electronic white-out. One can reconstruct previous states by undoing multiple actions. With selective undo it is also possible to reach novel states by redoing only a subset of actions performed from some previous state. This is like deleting a character in a word processor other than the one most recently typed. I am no longer so impressed.

We would like to go beyond the mindset of using recorded information to fix mistakes. Perhaps we can infer the user's intentions, or link interface actions to a broader range of related events similar in time or topic. In the context of Exploratory Data Analysis systems, we can treat the record of the exploration process as a dataset itself. This may benefit the analyst in remembering a train of thought, or by providing high level summaries of progress. It may also be used to familiarize coworkers with recent progress, for training students, or for presentations of results and justifications to management.

Our research group is pursuing a range of research into the uses of electronically captured experience. Hill and Hollan [1] were perhaps the first to point out the rich uses for history recording in a user interface. Programming-by-demonstration is an entire field based on this idea. In this paper, we focus on the idea that the data exploration process is not characterized by monotonic progress towards a goal, but rather involves much backtracking and opportunistic goal revision. An undo interface where actions are modeled as a linear sequence fails to capture this structure. A system that recognizes the structure and presents it visually potentially offers better support to the analyst for the purposes mentioned above. The same structure supports analyses that explicitly consider multiple alternative possibilities. Users of simulation packages often create such sets of scenarios by varying input parameters for multiple simulation runs.

In the next section we more fully describe previous undo models and our alternative "time travel" conception. We offer a principled architecture for capturing user actions and representing application states as *contexts*. We then describe the particular data exploration environment that our time travel interface runs in. The interface is based on a tree-structured visualization of scenario branching structure and also shows the actions performed in the current scenario. An example is given to illustrate its use. We then discuss capturing user intention along with actions and the improvement to selective redo this provides. Finally we present related work.

High-resolution color versions of all figures are available as, e.g., http://www.cs.cmu.edu/~sage/papers/KBS/Fig1.GIF. There is a shockwave animation of the interface at http://www.cs.cmu.edu/~sage/animations/TimeTravel.html.

2. Scenarios, Undo, and Context

The ability to quickly explore multiple scenarios is an important component of exploratory data analysis. For example, a financial spreadsheet can show the effect of alternative projected interest rates on profitability. For each interest rate, the user can browse and visualize various contributors to profit. These explorations are conceptually sequential operations, each building on the previous ones. Resetting the interest rate is best conceptualized as starting a new scenario that branches off from the previous one at a point before the rate-specific operations were performed. The entire exploration session might be summarized by graphing interest rate vs. profit for all scenarios in a single visualization.

One way to navigate among scenarios is with undo and redo. Previous undo implementations have concentrated on allowing undo or redo of single interface events, using buttons or menus. The simplest mechanism allows undo of only the most recently done, but not already undone, event. Selective undo allows the user to choose any done, but not already undone, event. This is usually called non-linear undo, because at any point the user has multiple choices of what event to undo. Due to possible confusion of non-linear with explicit representation of branching, we will use "selective" or "non-selective," and "branching" or "non-branching." There are complications and alternative semantics that have been proposed for selective undo [2], but there is no space to describe them here.

Most undo models represent history as an ordered list of events. An alternative, closer to the way we think about the real world, is that time is the organizing principle and events are landmarks on the real-valued time line. Following Rekimoto [3], we call continuous undo "time travel." In this view selective undo/redo involves replaying *intervals* of time within or across scenarios.

Several previous papers have commented on the possibility of showing the user a scenario tree, and point out that their semantics can handle branching [2, 4]. US&R [4] even maintains a graph-structured history internally. But no previous interface has exposed the conceptual branching structure to the user. Thus it has been very difficult to quickly navigate among the final states of all the scenarios, for instance, because the user is responsible for picking these states out from an undifferentiated menu.

Even if these states could be found, the user cannot analytically compare them. Previously interfaces can restore previous states by performing events' undo and redo methods, but at any moment they are in exactly one state. In order to compare scenarios, the user must rely on memory or make a copy of the system in different states [5]. There is no way to write a

formula that computes the difference in profitability of two scenarios with either the undo method or the copy method. And no previous system can create the interest rate vs. profit graph mentioned above.

In Artificial Intelligence, contexts are used as a convenient way to allow a single database at a single time to capture multiple states of a system. Each context must be self-consistent, but different contexts need not be mutually consistent. Using formulas that mention contexts explicitly, it is possible to compare quantities in different contexts, as in the Situation Calculus [6].

A major advantage of the context approach is that it is largely invisible to the underlying domain model. Formulas encoding domain constraints usually hold across all contexts, and need not mention context explicitly. For instance, profit is defined the same way in all contexts in terms of revenue and cost. The formula is always evaluated in a specific context, which provides values for the variables revenue and cost. The formula

$profit = revenue - cost$

implicitly stands for a schema that applies in all contexts

$\forall c\ profit_c = revenue_c - cost_c$

Without abstracting context this way, all the formulas in the domain model would have to incorporate extra variables. No previous GUI has used contexts to support undo.

3. Visage and Information-Centricity

Before describing the context model and time travel interface, for concreteness we describe the operations that support exploration within scenarios. By capturing these operations, we can support time travel. We used the Visage data visualization system [7], developed jointly by CMU and Maya Design Group. Visage is an *information-centric* [8] user interface environment for data exploration and for creating interfaces to data-intensive applications. Domain data objects are represented as first class interface objects that can be manipulated using a common set of basic operations, such as drill-down and roll-up, drag-and-drop, copy, and dynamic scaling. These operations are universally applicable across the environment, whether graphical objects appear in a hierarchical table, a map, a slide show, a query, or other application user interface. These containers are called frames. A frame is analogous to a window in other systems, and serves to visually organize logically related information. The most salient difference between windows and frames is that the latter can be nested.

An object-oriented database contains the complete state of the interface, as well as all domain data. This was originally done to allow multiple remote users to share information and visualizations. We capitalize on the explicit interface representation to reconstruct multiple contexts.

Three of Visage's extensional query operations are used in this paper: A user can *navigate* from the visual representation of any database object to other related objects. For instance in Figure 1 (1), the user has navigated from an army corps unit to its subordinate units. Graphical objects may be *brushed* [9] with a choice of colors, in which case graphical objects in all visualizations representing the same data object are colored identically. This kind of coordination enables the user to see correlations among more variables than can be encoded in a single visualization. In this paper, brushing is only used as a selection mechanism. In Figure 1 (3), the user has used a bounding box to brush the subordinates of the 53rd Mechanized Division, turning them all medium gray. *Dragging* or *copy-dragging* any of them to the bar chart brings along the whole identically colored set. (Subsequent brushing with a different color has changed some of them to lighter gray.)

<Insert Figure 1 about here.>

4. Branching Time Model

An exploration session begins in some initial database state. The analyst will explore for a while, and then backtrack and try an alternative scenario. Additional scenarios can branch off of the main trunk, or any previous branch scenario. We consider every tree branch to be a distinct scenario, and branches are created every time the user backtracks in time and performs new operations, as discussed in Section 6. Each scenario has a name, which is initially generated automatically. The root is labeled 1, followed by 1.1, 1.2, 1.2.1, etc. Users are encouraged to change the scenario names to something meaningful. The *x*-coordinate in this tree visualization represents time, and is labeled below the timeline.

A context is a <scenario, time> pair. Each time an update is made to the database, it happens in exactly one context. To organize the contexts into a tree, *lifting rules* are defined so that assertions made in one context affect other contexts [10]. Our only lifting rule is the following: The propositions that hold in a context are those that result from starting in the initial database state, traversing the unique path to the context and performing each recorded update. Note that some of these operations will have been done at an earlier time in the same scenario, while others will have been done in a parent scenario. Archer [11] calls this semantics for selective undo the "script model."

In order to implement contexts in Visage, we added an additional argument, context, to each database call. We modified the most general frame type to deal with contexts, so all frames inherit the capability. Any frame can be explicitly assigned a context. This can be done with the time travel interface or programmatically. If a frame has not been assigned an explicit context, it inherits that of its parent frame. The top-level frame always has the context <1, now>, that is, the root scenario and the current time. "Now" is a special symbol that may be used instead of an absolute time and is reevaluated each time it is

accessed. When an interface object sends a message to query or update the database, its parent frame handles the message and adds its associated context before calling the database function.

The propositions that hold in <1, now> are cached. For all other contexts, a database query involves walking backward from that context until an answer is found. This implementation does not impose a penalty unless the user is time traveling, in which case the cost of querying is linear in the number of database updates. The CSpace project, which is developing sublinear algorithms for querying multiple versions of documents and data, demonstrated fast response given 10,000 versions about 500,000 objects [12]. If applied to Visage this would amount to 10,000 reachable contexts, where each interface event would require one context.

A changeContext command was added to the database API. When a frame changes its context, its appearance must reflect the new state. It is given a package of updates optimized to include only true differences between the old and new contexts, rather than every update that occurred on the path between them. For instance, if a bar chart is created and then deleted, the containing frame will not have to waste time repeating those operations. The frame processes the changes just as it does for real-time updates. The complexity of this operation is also linear in the number of database updates.

Finally, a holdsIn operator was added to the attribute definition language to allow explicit control of context. Consider a base interest rate scenario of 4.0%, which we want to compare with several other scenarios. We can define

deltaProfit = *profit* – holdsIn(baseScenario, *profit*)

The new attribute will have the appropriate value in each context.

5. Time Travel Interface

Figure 2 shows three visualizations showing similar information to Figure 1. Below them is the TimeTravel interface, showing the events described in Figure 1. The annotations above events correspond to the numbers in Figure 1. By default, Visage creates events for drag, navigation, and brushing operations. In addition, the user can create events with explicit begin and end gestures. In Figure 2, the user has grouped some related low-level events with a manually created event labeled "Locate high supply units." An event is always considered a subevent of the lowest level current event. All events in an exploration session will be descendents of the event "Visage session." Thus hierarchical events are created automatically. The full names of the events, which can only be seen in other interfaces, are

Drag ARMY
Nav >subord_unit
Nav >subord_unit
Select echelon

Drag 6 Units
Select total_supply_weight
Drag 2 Units

<Insert Figure 2 about here.>

Below the events is a degenerate tree showing the root branch, as well as a linear timeline. The I-beam sliders in the tree or the timeline are used for time travel. As the sliders move, the frame's context changes and the visualizations show an animation of the event history. Here, the user has traveled backward from the state shown in Figure 1 to a point just after the first navigation event. Scenario-based time travel navigation can be accomplished by clicking anywhere on the tree, or by dragging the slider anywhere on the tree. Alternatively, chronological time travel can be accomplished by dragging the slider on the timeline below the scenario tree. The system then chooses the scenario that was active at that real time. The active scenario is defined to be the one in which the most recent database update was made. Tight coupling between the context tree and chronological sliders maintains consistent positions no matter which one is dragged.

6. Acting in Time

User actions performed in a context whose time is "now" are recorded in the current scenario and time-stamped with the real time. In the more complicated case, the user moves a slider to a context in the past before taking an action. In that case a new scenario is created and the context is set to <new scenario, now> before the action is performed. Thus operations are always performed in the present. This avoids problems that may arise when temporally extended events are selectively redone. The events might overlap other events in the dragged-to scenario, or might extend into the future. It also avoids any possibility of time travel paradoxes. However the fact that we never change the past of any scenario, instead creating new ones, also avoids these paradoxes. In addition, all information about history and branching are recorded in <1, now>, and all TimeTravel interfaces operate in this context. If the interface changed its own context, it would be impossible to use it to time travel to other scenarios.

In Figure 3, the user has performed an alternate navigation from the state shown in Figure 2, showing the subordinates of the 22nd Division rather than the 53rd Division. Since this new operation is performed in the past, a new branch, labeled 1.1 is created. It becomes the current branch, and is shown in black. Other branches are shown in gray. The line width shows which scenario was active at a given time. Only the events performed in the current branch are visible. Line widths and events shown are updated on mouse release. A previous version of the TimeTravel interface rearranged tree branches so that the current branch was always on top, next to its events. However testing showed that this confused users.

<Insert Figure 3 about here.>

Selective redo is performed by drag-and-drop. Any subset of events can be selected and a copy dragged to any point on the context tree. In Figure 4, the user has traveled back to the first branch, in order to see its events. The user selects "Locate High Supply Units" and drags it to "now" in branch 1.1 (arrow).

<Insert Figure 4 about here.>

If the event had been dropped in the past, a new branch would have been created. Since it is dropped "now," the database updates that occurred during the event are applied directly in 1.1 (see Figure 5). In either case, the timestamp recorded for the updates is the current real time. Then any updates that follow the drop point in the dropped-on scenario (and any later branches off this scenario) would be applied, also timestamped with the real time. Since replaying each event takes finite time, the timestamps of successive events differ slightly. Thus the temporal order is maintained, although the quantitative relationships are lost. This sacrifice is made so that formally all updates happen "now," even though conceptually we are updating in the past. Comparing Figures 4 and 5 shows the difference between the final states of scenarios 1 and 1.1. To perform this comparison statically, the frame (not shown in figure) enclosing the visualizations and TimeTravel interface (in scenario 1) can be copied. Its context can then be set to scenario 1.1, allowing side-by-side comparison.

<Insert Figure 5 about here.>

Any subset of events can also be selectively undone by dragging out of the time travel interface. A new scenario is created at the beginning of the earliest selected event, and any remaining events are replayed.

Object identity is shared by all contexts, and it is always possible to make arbitrary database updates. An object that "does not exist" in a context simply has no attributes there. However, with selective undo/redo, some database updates may not have a useful effect. In the "Locate high supply units" example, the dragged event would have no useful effect because it operates on the units in the 53rd Division. Thus nothing would be brushed or dragged using simple replay of database updates.

7. Inferring Intention

By inferring the user's intention we can do a better job of replaying events in a different scenario. We now describe the two simple heuristics we use to infer user intention and generalize recorded events.

Visage's basic operations of brushing, drag-and-drop, and navigation record information about how the user performed the operation, as well as the low-level database updates that result. For navigation and dragging, the brushing color of the dragged object is recorded along with the identity of the object and the source and target frames. If the object has no brushing

color, replay of the event is applied to the same object. Otherwise, the object identity is ignored and the event is applied to any objects in the source frame with the recorded brushing color.

In general we try to use very simple heuristics like this that the user can readily understand and predict. Not only are they a natural interpretation of the user's intention, but the user can purposely use them to signal intention. Associating colors with macro arguments is such a cheap operation that we hope users will get in the habit of using it automatically. More intrusive interfaces for recording intention at execution time are rarely used.

In Visage, brushing is accomplished either by clicking on a single object or with a bounding box around multiple objects. In the former case, we offer no heuristics for inferring intention. In the latter case we infer that the intention is to include any objects that would fall in this region of the visualization. For instance the horizontal extent of the bounding box in the bar chart of Figure 1 extends from 150 tons to 1800 tons, so we infer that the intention is to pick out objects whose supply weight falls in this range. Since the box extends above and below the range of the *y*-axis, we infer that the user does not intend to take the name attribute into account when choosing which objects to brush. Again, we hope that this distinction is so transparent that the user will get in the habit of purposely drawing bounding boxes that include the whole *y*-axis range, even if a smaller box would enclose all the same objects for the data in the current scenario.

Having used bounding boxes to select objects (Figure 1), the selectively redone event in Figure 4 is replayed at the level of intentional brushing and dragging. Thus the user has successfully used the exploration sequence performed on the 53rd Division to see an identical analysis of the 22nd Division. Copying the manually created high-level event "Locate High Supply Units" to the new scenario would be a useful additional feature.

8. Future Work

8.1. Temporal Domains

When exploring temporal domains, it is convenient to have the environment transparently perform the temporal reasoning. For instance, when planning budgets over several years, one may want updates to one year to propagate to succeeding years until explicitly changed. Encoding defaults like this using spreadsheet macros would vastly complicate the model. Further, the time travel interface is much easier to use than anything that could be programmed into the spreadsheet.

Using contexts for both undo and domain time travel requires two distinct times to be recorded for each database update: the real time when the interface action is performed, and the simulated domain time when data updates take effect. The lifting rule changes slightly: The propositions that hold in a context are those that result from starting in the initial database state,

traversing the path to the context's scenario and domain time and performing each recorded update *whose user timestamp is at or before the context's user time*. Updates are ordered primarily by domain time, and secondarily by user time.

The travel interface would have a domain-time slider and a user-time slider in the context tree and on the timeline. The frame's context is a function of both: <scenario, user time, domain time>. Changing the scenario of either slider would propagate to the scenario of the other. Without maintaining both sliders, there would be no way to distinguish whether the user wants to see the current version of last year's budget, or last year's version of this year's budget. The asymptotic complexity of time travel is still linear in the number of events. User testing will clarify whether maintaining two times is too confusing. No previous interface has made the attempt.

8.2. Superimposed Contexts

In some situations it would be useful to visualize a range of contexts simultaneously. For instance the scatterplot of quarterly profit over all the interest rate scenarios mentioned in Section 2 shows the predicted range that can be expected. The scatterplot expects to show one plot point for each budget object that it is told to display. But there is only one budget object in the whole database, whose attributes change with time and scenario.

In situations like this it is easier to reason about reified states of objects. On an as-needed basis, we would create *temporal subabstraction* [13] objects like Q3_budget_4%scenario. These objects would have fixed profit values in all contexts. Many such objects could then be displayed simultaneously in the plot chart. Any applicable database update to the budget object would be reflected in the temporal subabstraction. Updates to the temporal subabstraction are just a shortcut for updating the budget object in Q3, 4%scenario.

9. Related Work

9.1. Undo

Myers and Kosbie [14] introduce an elegant mechanism for hierarchical undo. Each interface action is given a DO and UNDO method, and a pointer to its parent action, if any. This allows the user to undo an entire semantic action, or just the subactions (e.g. a dialog box action). By adding the context layer to the database, Visage automatically captures the state changes caused by any action. The application developer does not need to write DO and UNDO methods. Hierarchical undo merely requires that the actions issue "begin event" and "end event" messages. In order to capture intention and perform more intelligent selective redo, however, we are using exactly the mechanism of Myers and Kosbie.

9.2. Macro Definition

Chimera uses a comic-strip visualization to show successive states of an interface [15]. Chimera is clever about generating comic strip panels, which show only the subset of the screen relevant to the current event. The user can change the granularity of the comic strip, analogous to looking at the different rows in Figure 2. Chimera histories are linear. Showing a comic-strip panel over each visualized event would enhance their recognizability in the TimeTravel interface.

Chimera allows macro definition from the history of interface actions. The user selects a subset of events to generalize into a macro. A macro builder window pops up containing a comic strip of these events. The user then selects arguments to the macro graphically, and generalizes the macro to apply in a variety of situations. Chimera has an inference engine to guess default generalizations, which the user can override by selecting from a list of possible alternatives.

The explicit user interaction at macro-building time would be a valuable complement to the simple heuristics we use at the time events are recorded. Macros could be visualized in a separate row of Figure 2. To apply them, they could be dragged to other points on the context tree just as in selective redo.

We have explored an alternative to macros, creating persistent *information appliances* [7] from recorded events. Appliances are custom user interfaces for performing a specialized class of tasks. They are declarative alternatives to traditional procedural macros. Unfortunately there is no space to describe them here.

9.3. History Enriched Objects

Hill and Hollan [1] describe a set of applications enhanced to show information about their usage history. For instance Edit Wear displays a histogram of the number of times each line in a file has been edited. The histogram is shown in the scrollbar. Spreadsheet Wear colors the background of cells to show the number of recalculations. The lines and cells are called "history-enriched objects."

The same mechanism that supports time travel in Visage also supports history-enriched objects. Each time a Visage database update occurs, the context layer of the database creates a history object. This object records the user, domain object, attribute or relation, new value, scenario, and the real time of the update. The history objects are first-class and can be visualized as easily as any other object. For instance, a bar chart might show the number of times each person's salary has been edited during the budget creation process. Thus all Visage objects are history-enriched.

However, Visage's history mechanism is rather clumsy for text editing, as it views any edit as completely changing the value. Better would be a difference analyzer so database updates are deltas linked to specific lines of text. Given this, it

would be straightforward to add custom scrollbar histograms to the Visage editor frame. The more interesting capability Visage provides is allowing the end user to create custom displays of usage history on the fly.

9.4. Timeline Interfaces

In addition to Chimera, described above, there have been several interfaces that rely on timeline visualizations for undo, browsing and/or time travel.

TimeScape [3] uses a timeline metaphor to organize objects on the computer desktop, rather than the usual folder hierarchy. The existence and position of the objects is recorded. A timeline slider restores earlier states of the desktop's appearance. A document created in the future becomes a reminder, as it pops into existence at the appointed time. There is an alternate viewing mode where the plane of the desktop is swept left to right leaving tracks along the third dimension. Undo/redo is non-selective.

Interlocus [5] takes discrete snapshots of workspaces, allowing the user to return to previous appearance states, providing non-selective undo/redo.

Lifelines [16] is a timeline based visualization of personal histories. It is oriented toward real life events such as medical histories. It presents a single broad overview of many events and attributes and allows easy filtering and drill down to detailed information. Its pan and zoom features would be a helpful addition to Visage's time travel interface. Unfortunately real life events cannot be undone in our universe, nor can we try out alternate scenarios. Therefore the interface does not support these features.

Learning Histories [17] extend the interface of Lifelines to physical simulation programs where undo and modification is possible. In their example, training operators when to turn on and off valves to a vacuum pump, the relative timing between events is crucial. When modifying the history, the original timestamps are therefore maintained. Branching histories are not supported. It would be interesting to combine maintaining timestamps with branching history.

Meng et al [18] visualize a linear sequence of snapshots showing the resulting state after each interface event. The user chooses an event to undo/redo from the visualization, rather than from a text menu. The most interesting contribution is the ability to filter the visualization to only show events that affect a specific object. The user can also "flash" all the events of a certain type, such as create, move, or change color. These capabilities make it easier to find the desired event to undo.

Although not specifically designed to support these filtering and selection operations, the Time Travel interface intrinsically provides this capability. As in any Visage frame, Dynamic Query [19] widgets can be added and set to filter the interface events by type. A visual query [20] linking events to their history objects would allow DQ filtering to display only

events having at least one history object mentioning a given domain object. Events from the TimeTravel interface could even be copied and dragged to a scatterplot organizing them by, say, frame and event type as well as filtered by time. Then a selected subset could be dragged back to a point on the context tree to effect selective redo.

9.5. Collagen

Collagen shares our goal of enabling "better support for the intentional level of human-computer interaction, especially as it unfolds over time" [21]. Their approach is based on discourse theory, and provides a higher-level view of user interaction than the basic operations described here. We have the infrastructure to construct hierarchical histories, but not the discourse and domain modeling necessary to extract structure with minimal user involvement. Testing has shown that it is difficult for users to relate the atomic operations shown in the TimeTravel interface to their intentions. Adding intelligence like Collagen's is a promising means for bridging this gap. Collagen's history structure is limited to hierarchies of predefined recipes, and is displayed using indented text. Users can revisit previous recipe instantiations and achieve their goals in a different way. Then subsequent recipe instantiations can be replayed to adapt to the changed context. This is more flexible than non-selective undo, but still restricted by the fixed hierarchical structure. Changing the structure loses the history information. Branching scenarios are not supported.

10. Contributions

Visage's time travel implementation combines ideas from diverse research areas in a new way. Like CSpace it manages a branching version structure. Using the formalism of contexts, it goes beyond the ability to be "in" any of the versions: the holdsIn operator allows analytical comparisons across contexts as well. Different Visage frames can be in different contexts, so visual comparison is also easy. Since context-specific updates are captured automatically by a layer built on top of the database, Visage is the first system to support undo without requiring application developers to write explicit undo methods

Visage includes the first visualization-based interface to a branching scenario structure, using ideas from the non-branching timelines of TimeScape, Interlocus, LifeLines, and Chimera. It is the first to use drag-and-drop selective undo/redo for easy manipulation of multiple events at once. Due to the universal applicability of Visage's basic operations, sophisticated filtering and selection of events to undo is intrinsic.

Acknowledgements

This work was supported by DARPA contract DAA-1593K0005. Stephan Kerpedjiev suggested helpful modifications to the design of the interface and the presentation of this paper.

References

1. Hill, W.C. and J.D. Hollan, *History-Enriched Digital Objects: Prototypes and Policy Issues.* The Information Society, 1994. **10**: p. 139-145.

2. Berlage, T., *A Selective Undo Mechanism for Graphical User Interfaces Based on Command Objects.* ACM Transactions on Computer-Human Interaction, 1994. **1**(3): p. 269-294.

3. Rekimoto, J. *TimeScape: A Time Machine for the Desktop Environment*. in *Human Factors in Computing Systems (SIGCHI)*. 1999. Pittsburgh, PA: p. 180-181.

4. Vitter, J.S., *US&R: A New Framework for Redoing.* IEEE Software, 1984. **1**(4): p. 39-52.

5. Hayashi, K. and E. Tamaru. *Information Management Strategies Using a Spatial-Temporal Activity Structure*. in *Human Factors in Computing Systems (SIGCHI)*. 1999. Pittsburgh, PA: p. 182-183.

6. McCarthy, J., *Programs with Common Sense*, in *Readings In Knowledge Representation*, R.J. Brachman and H.J. Levesque, Editors. 1985, Morgan Kaufmann. p. 299-308.

7. Kolojejchick, J.A., S.F. Roth, and P. Lucas, *Information Appliances and Tools in Visage.* Computer Graphics and Applications, 1997. **17**(4): p. 32-34. http://www.cs.cmu.edu/~sage/PDF/Appliances.pdf.

8. Roth, S.F., M.C. Chuah, S. Kerpedjiev, J.A. Kolojejchick, and P. Lucas, *Towards an Information Visualization Workspace: Combining Multiple Means of Expression.* Human-Computer Interaction Journal, 1997. **12**(1-2): p. 131-185. http://www.cs.cmu.edu/~sage/PDF/Towards.pdf.

9. Becker, R.A. and W.S. Cleveland, *Brushing Scatterplots.* Technometrics, 1987. **29**(2): p. 127-142.

10. Guha, R.V., *Contexts: A Formalization and Some Applications*, 1991, PhD thesis, Stanford: Palo Alto. 147 pages. http://www-formal.stanford.edu/guha/guha-thesis.ps.

11. Archer, J.E., R. Conway, and A.J. Dix, *User Recovery and Reversal in Interactive Systems.* ACM Transactions on Programming Language Systems, 1984. **6**(1): p. 1-19.

12. Scherlis, W.L., *Asynchronous collaboration services for single-user applications using situated events and structural models*, CSPACE Project unpublished working paper. 1999, Carnegie Mellon University: Pittsburgh, PA.

13. Guha, R.V. and D.B. Lenat, *Cyc: A Mid-Term Report.* AI Magazine, 1990. **11**(3): p. 32-59. http://www.cyc.com/tech-reports/act-cyc-134-90/act-cyc-134-90.html.

14. Myers, B.A. and D.S. Kosbie. *Reusable hierarchical command objects.* in *Human Factors in Computing Systems (SIGCHI)*. 1996. Vancouver, BC, Canada: ACM Press: p. 260-267.

15. Kurlander, D. and S. Feiner. *A History-Based Macro by Example System*. in *User Interface Software and Technology (UIST)*. 1992. Monterey, CA: ACM Press: p. 99-106.

16. Plaisant, C. and B. Shneiderman, *An Information Architecture to Support the Visualization of Personal Histories.* Information Processing & Management, 1998. **34**(5): p. 581-597. ftp://ftp.cs.umd.edu/pub/hcil/Reports-Abstracts-Bibliography/3855HTML/3855.html.

17. Plaisant, C., A. Rose, G. Rubloff, R. Salter, and B. Shneiderman, *The Design of History Mechanisms and their Use in Collaborative Educational Simulations*, 99-11. 1999, University of Maryland Human Computer Interaction Laboratory. http://www.cs.umd.edu/hcil.

18. Meng, C., M. Yasue, A. Imamiya, and X. Mao. *Visualizing Histories for Selective Undo and Redo*. in *Third Asian Pacific Computer & Human Interaction*. 1998. Kangawa, Japan: IEEE: p. 459-464. http://dlib.computer.org/conferen/apchi/8347/pdf/83470459.pdf.

19. Ahlberg, C., C. Williamson, and B. Shneiderman. *Dynamic Queries for Information Exploration: An Implementation and Evaluation*. in *Human Factors in Computing Systems (CHI)*. 1992. Monterey, CA: ACM Press: p. 619-626.

20. Derthick, M., J.A. Kolojejchick, and S. Roth. *An Interactive Visual Query Environment for Exploring Data*. in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*. 1997. Banff, Canada: ACM Press: p. 189-198. http://www.cs.cmu.edu/~sage/UIST97/UIST97.pdf.

21. Rich, C. and C.L. Sidner. *Segmented Interaction History in a Collaborative Interface Agent*. in *Proceedings of the Intelligent User Interfaces Conference(IUI)*. 1997. Orlando, FL: p. 23-30.

Figure Captions

Figure 1 Illustration of direct manipulation operations in Visage. 1) Navigate (drill down) from an Army corps to its four constituent divisions (first level of indentation). 2) Navigate from 53rd mechanized division to its constituent brigades. These are then selected using brushing with a bounding box, and 3) a copy is dragged to the bar chart, where the same units are represented in a different way, showing the weight of their supplies. 4) The units requiring the most supplies are selected with a bounding box, and a copy is dragged to the map.

Figure 2 TimeTravel interface showing events of Figure 1, most of which have been undone.

Figure 3 Actions performed in the past generate new branches.

Figure 4 Return to branch 1.

Figure 5 Effect of selective redo. The redone operations all occur in the short interval required for their execution now.

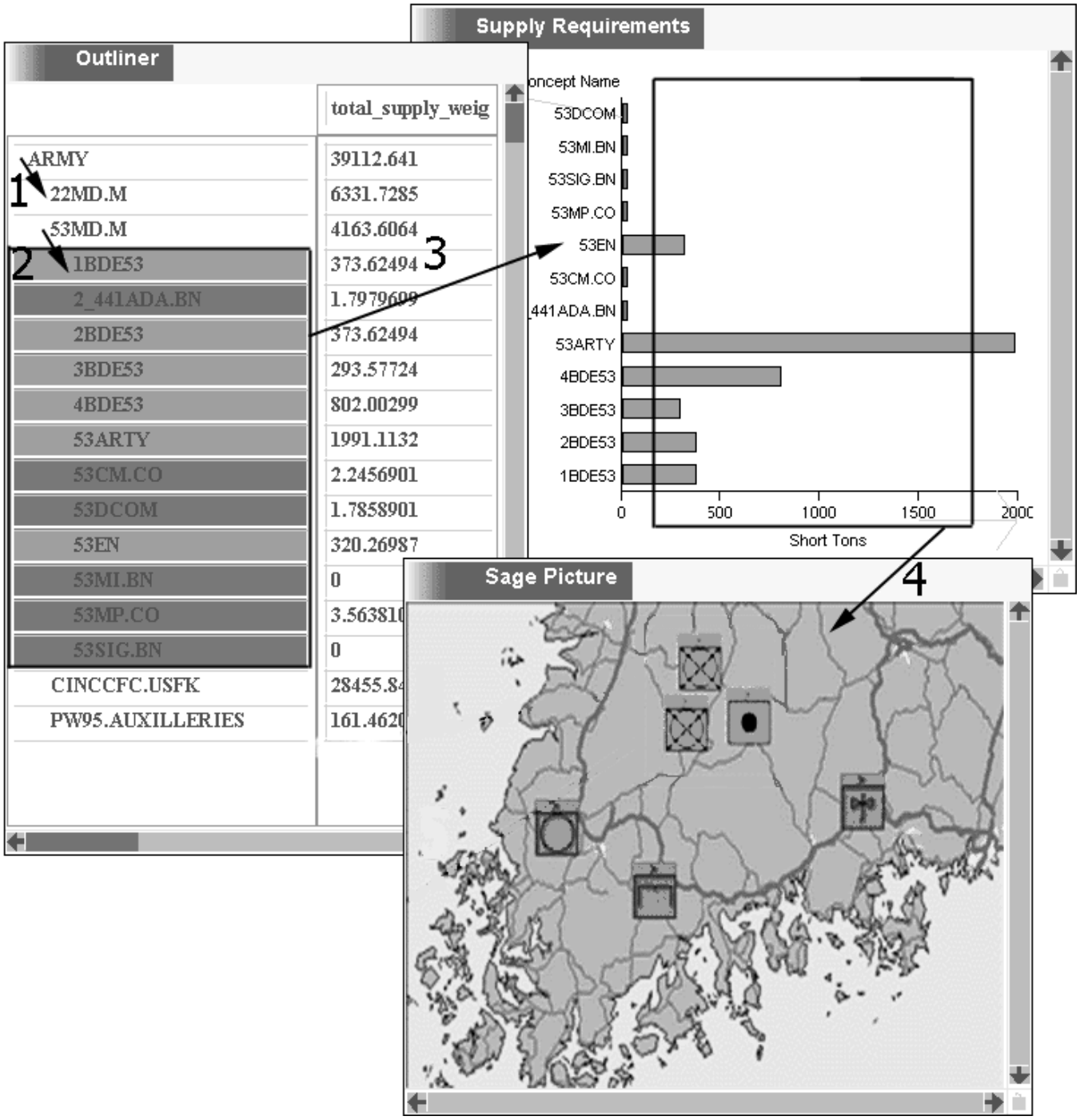
Outliner
total_supply_weig
ARMY
22MD.M
53MD.M
1BDE53
2_441ADA.BN
2BDE53
3BDE53
4BDE53
53ARTY
53CM.CO
53DCOM
53EN
53MI.BN
53MP.CO
53SIG.BN
CINCCFC.USFK
PW95.AUXILLERIES
39112.641
6331.7285
4163.6064
373.62494
1.7979699
373.62494
293.57724
802.00299
1991.1132
2.2456901
1.7858901
320.26987
0
0
28455.84
161.4620
1
2
3
4
Supply Requirements
oncept Name
53DCOM
53MI.BN
53SIG.BN
53MP.CO
53EN
53CM.CO
441ADA.BN
53ARTY
4BDE53
3BDE53
2BDE53
1BDE53
0
500
1000
1500
2000
Short Tons
Sage Picture

Figure 1

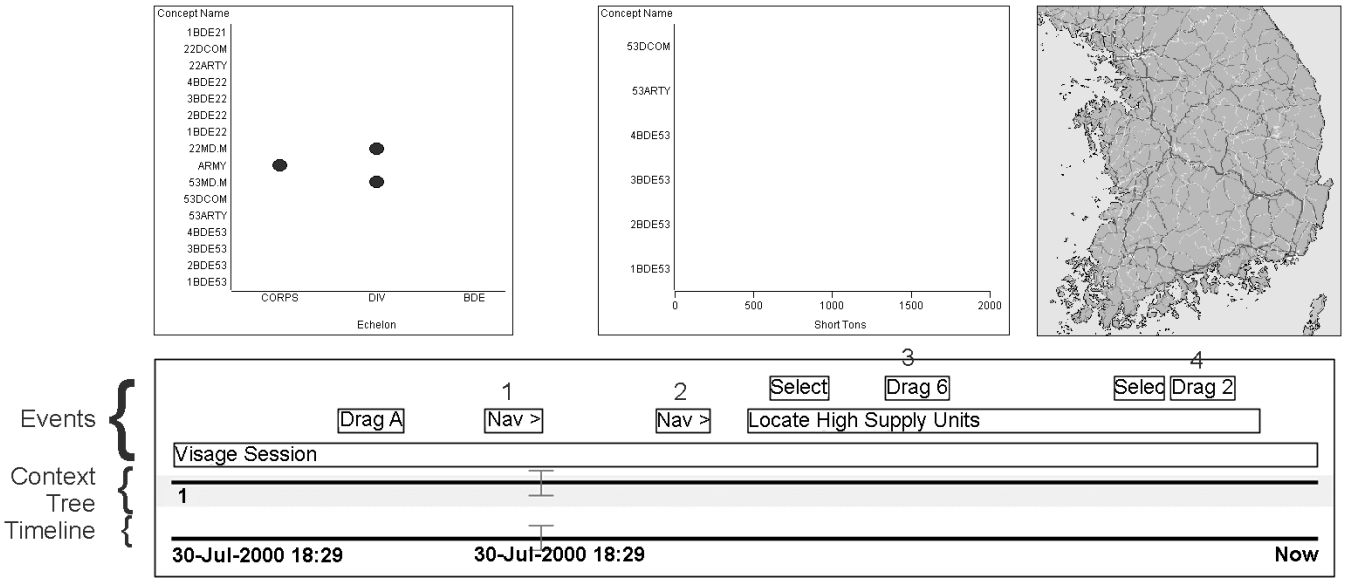
Concept Name
1BDE21
22DCOM
22ARTY
4BDE22
3BDE22
2BDE22
1BDE22
22MD.M
ARMY
53MD.M
53DCOM
53ARTY
4BDE53
3BDE53
2BDE53
1BDE53
CORPS
DIV
BDE
Echelon
Concept Name
53DCOM
53ARTY
4BDE53
3BDE53
2BDE53
1BDE53
0
500
1000
1500
2000
Short Tons
Events
Context
Tree
Timeline
1
2
3
4
Drag A
Nav >
Nav >
Select
Drag 6
Seled
Drag 2
Locate High Supply Units
Visage Session
1
30-Jul-2000 18:29
30-Jul-2000 18:29
Now

Figure 2

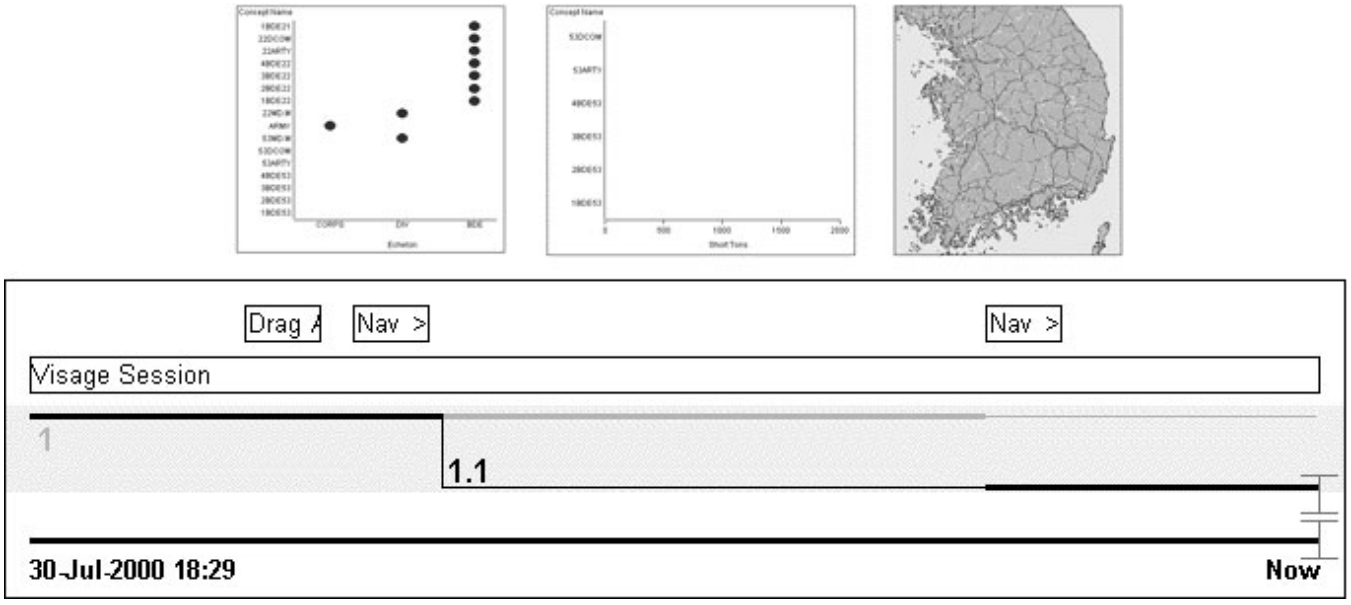

Figure 3

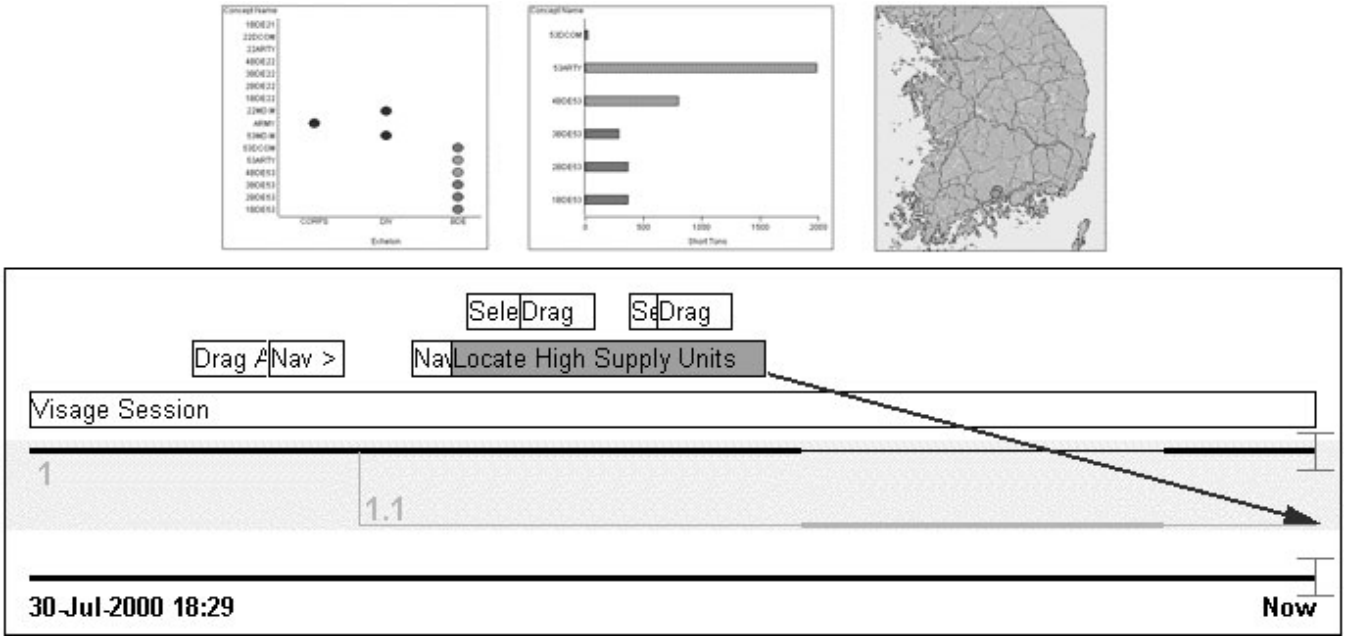
Sele Drag
Se Drag
Drag A Nav >
Nav Locate High Supply Units
Visage Session
1
1.1
30-Jul-2000 18:29
Now

Figure 4

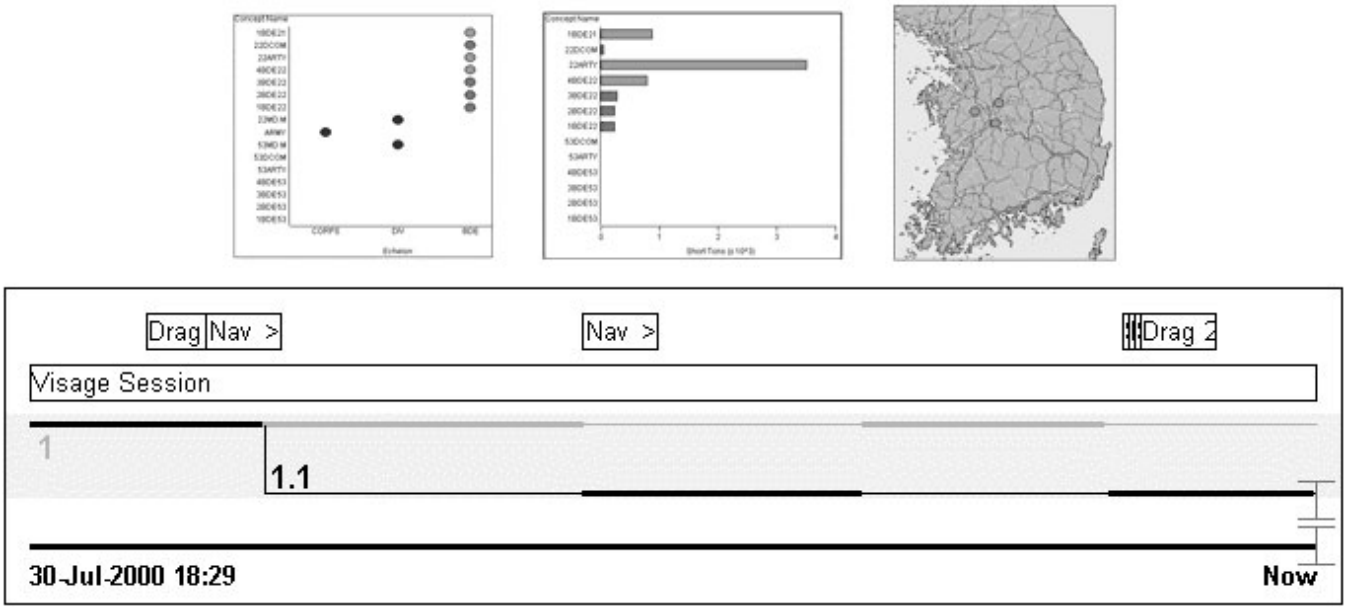

Figure 5