Proceedings of Intelligent User Interfaces (IUI '00), New Orleans, LA, January, 2000, pp. 60-67.

Data Exploration across Temporal Contexts

Mark Derthick and Steven F. Roth
Carnegie Mellon University Robotics Institute
{mad+, roth+}@cs.cmu.edu

Abstract

The ability to quickly explore and compare multiple scenarios is an important component 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 the user to use her 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 user 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 could treat the record of the exploration process as a dataset itself. This could benefit the analyst in remembering her train of thought, or providing high level summaries of her progress. It could 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. Users who can't visualize history structure are doomed to repeat it poorly. 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 a particular data exploration environment in which we've implemented our time travel interface. It is based on a tree-structured visualization of scenario branching structure and also shows the actions performed in each 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.

A first version of the interface has been implemented. This paper describes additional features, as noted in the text, which will be added to the second version. High-resolution color versions of all figures are available as, e.g., http://www.cs.cmu.edu/~sage/papers/IUI00/Fig1.GIF. There is a shockwave animation of the interface at http://www.cs.cmu.edu/~sage/papers/IUI00/demo.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 various 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 sequential 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], which there is no space to describe 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 time line. Following Rekimoto [3], we call non-selective undo 'time travel.' In this view selective undo 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 for the user to quickly navigate among the final states of all the scenarios, for instance, because he is responsible for picking these states out from an undifferentiated menu.

Even if he could find these states, he 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. 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 [5].

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

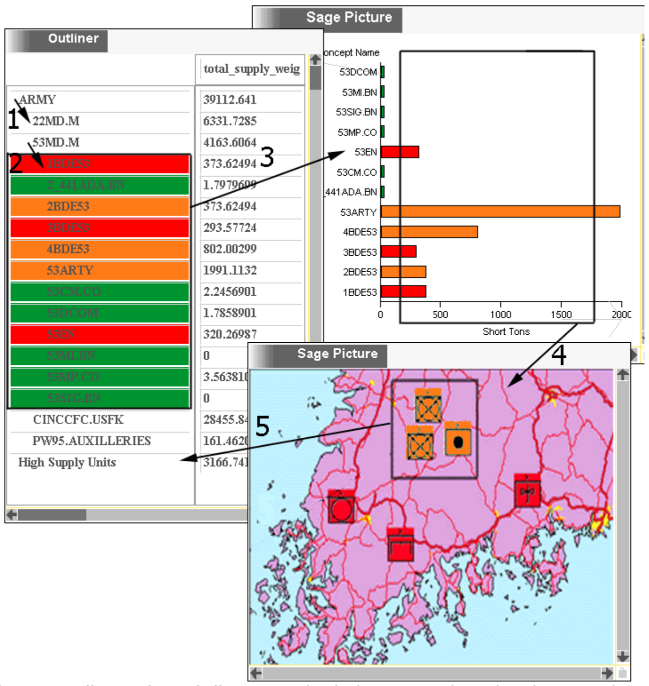
$$\forall c \; profit_c = revenue_c - cost_c$$

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 [6], developed jointly by CMU and Maya Design Group. Visage is an *information-centric* [7] 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.

Integrating the visualization system directly with an underlying database, rather than just deriving visualizations from individually exported tables, is key to coordinating visualization applications with the other components of an exploratory data analysis environment. For instance, graphical objects can be dragged across application UI boundaries. The integral database also made it easy to implement time travel using contexts, as described in Section 4.



	total_supply_weig
ARMY	39112.641
22MD.M	6331.7285
53MD.M	4163.6064
1BDE53	373.62494
2_441ADA.BN	1.7979609
2BDE53	373.62494
3BDE53	293.57724
4BDE53	802.00299
53ARTY	1991.1132
53CM.CO	2.2456901
53DCOM	1.7858901
53EN	320.26987
53MI.BN	0
53MP.CO	3.563810
53SIG.BN	0
CINCCFC.USFK	28455.84
PW95.AUXILLERIES	161.4620
High Supply Units	3166.741

Figure 1 Illustration of direct manipulation operations in Visage. 1) Drill down from an Army corps to its four constituent divisions (first level of indentation). 2) Drill down 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. Notice that as a result of brushing the background colors of units are coordinated between visualizations. 4) The units requiring the most supplies are selected with a bounding box, and a copy is dragged to the map. Here the units are represented by domain-specific icons showing their type and echelon. 5) Three of the high-supply units farthest inland are selected and aggregated. The aggregate, dubbed 'High Supply Units,' appears in the outliner, where properties like total supply weight can be shown.

Visage includes five ubiquitous extensional query operations: 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. He can *brush* [8] graphical objects in a visualization with a choice of colors, and have graphical objects in all visualizations representing the same data object also be colored. This kind of coordination enables the user to see correlations among more variables than can be encoded in a single visualization. In Figure 1 (3), the user has used a bounding box to brush the subordinates of the 53rd Mechanized Division, turning them all green. Dragging any of them to the bar chart brings along the whole similarly colored set. (Subsequent brushing with different colors has changed some of them to red and then orange.)

It is also possible to roll up, or *aggregate*, database objects into a new object, which will have attributes derived from its elements. In Figure 1 (5), the user has aggregated three high-supply units located near one another on the map. The aggregate appears in the outliner. He could then look up the total supply needs, or *partition* this set into sub-aggregates based on role (e.g., all medical units would form one element of the partition).

The user can *filter* objects based on any attribute with a Dynamic Query slider [9]. Graphical objects representing a data object whose attribute value falls outside the range selected by the slider are made invisible. Filtering is another way to coordinate multiple visualizations and see relationships among more variables than can be shown in a single visualization.

4. Branching Time Model

The user will begin his exploration session in some initial database state. He 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. This leads to the tree-structured branching time model illustrated in Figure 2 (middle section of large rectangle). 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 of the four scenarios shown has a name, which is initially generated automatically. The root is labeled 1, followed by 1.2, 1.2.2, and 1.3. 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 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. Only one piece of interface code had to be modified. 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 the current 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 [Bill Scherlis, personal communication 1999]. 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 won't 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 (middle section of large rectangle) shows how the interface arranges the scenarios. The current scenario for this frame's context, 1.3, is shown as the horizontal line that constitutes the top tree branch. The time is shown as a slider position (red rectangle) on the current scenario. Here the slider is at the rightmost time coordinate, representing 'now.'

As the user travels in time, the frame's context is continually changed. This generates an animation of its history. 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. The active scenario is shown with thicker line segments in the scenario tree and with labels on the chronological timeline. Tight coupling between the scenario and chronological

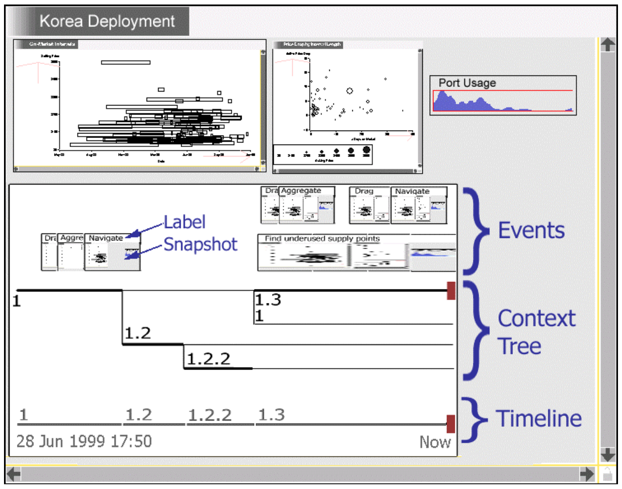


Figure 2 A Visage frame containing an interval chart, scatter plot, histogram, and time travel interface (large bottom rectangle). For expository purposes, the time travel interface has been made larger than would be typical, while the other subframes have been made smaller.

sliders maintains consistent positions no matter which one is dragged.

When time traveling to a new scenario, the vertical position of branches is minimally changed by moving other scenarios down. The change occurs on mouse release.

Interface events are shown for the current scenario using a comic strip notation [12] (see Figure 2). There is a label describing the event and a snapshot of the frame's appearance at the conclusion of the event[1]. By default, Visage creates events for drag, navigation, and brushing operations. In addition, the user can create events, defining the start and end. In Figure 2, the user has created an event labeled "find underused supply points."

Comic strips at different levels of detail are stacked, with the most detailed on top. For instance, "find underused supply points" is accomplished by two drags, a navigate, and an aggregate. An event is always considered a subevent of the most detailed current event in the current scenario, if any. Thus hierarchical events are created automatically.

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, avoiding paradoxes.

Selective undo/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. A new scenario is created at that point, and the database updates that occurred during

[1] The current implementation has only a label. The snapshots were added to the figures in this paper with Photoshop.

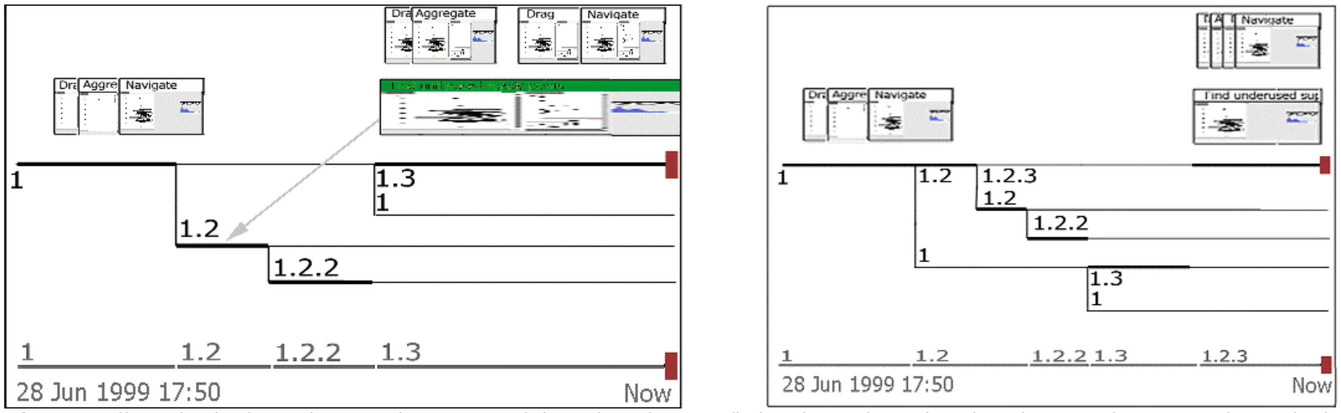


Figure 3 Effect of selective redo. Dragging a copy of the selected event "find underused supply points" in Scenario 1.3 to a time point in Scenario 1.2 (left) creates a new scenario, 1.2.3 (right). The new scenario is shown on top. The redone operations all occur in the short interval required for their execution 'now.'

the events are applied. The timestamp recorded for the updates is the current real time. Then the rest of the updates that follow the branch point are applied, also timestamped with the real time (see Figure 3). Since replaying each event takes finite time, the timestamps of successive events will 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.

Any subset of events can also be selectively undone by dragging them 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 "doesn't exist" in a context simply has no attributes there. However, with selective undo/redo, some database updates may not have a useful effect. For instance, redoing a drag into a frame that doesn't exist in some context will add a member relationship to the otherwise empty object. But since the object doesn't represent a frame in this context, the interface won't show a change.

7. Information Appliances by Demonstration

By inferring the user's intention we can do a better job of replaying events in a different scenario. For instance after the sequence performed in Figure 1, the analyst may want to look for divisions of a different army corps with high supply needs. The stored database updates contain information such as "add 53ARTY to the bar chart." Replay using this low-level representation of the drag event cannot be transferred to new data objects.

In previous work, recorded history has been used to create macros with explicit arguments [12]. The system generalizes the example to abstract out the macro arguments, either heuristically or through a dialog with the user. In our example, if the corps becomes an argument we can meaningfully replay the macro in a scenario analyzing a different corps.

We now describe the two simple heuristics we use to infer user intention and generalize recorded events. We then describe how we can use intention to do a better job of selective redo. Visage can also create persistent *information appliances* 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.

7.1. Inferring Intention

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 his 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 concept 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.

7.2. Selective Redo

When the user drags events to a point on the scenario tree, as in Figure 3, we make use of any inferred intention rather than replaying the low-level database updates by rote. Iterating through the dragged events in temporal order, we send each one a message to redo itself. If it is one for which we've captured the intention, it uses a hand-written redo method. Otherwise it defaults to iterating over its subevents, recursively telling them to redo themselves. The default case also uses the changeContext mechanism described in Section 4 to effect database updates that are their immediate children.

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 wants 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 that two distinct times be recorded for each database update: the real time at which the user performs the interface action, and the simulated time at which he wants any domain data updates to 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 and performing each recorded update *whose user time is at or before the context's user time.* Updates are ordered primarily by domain time, and secondarily by user time.

Each frame can have two time travel interfaces. One is for user time and the other for domain time. The frame's context is a function of both interfaces: <user scenario, user time, domain scenario, domain time>. Without maintaining both interfaces, 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'd 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 he have the actions issue 'begin event' and 'end event' messages.

In order to capture intention in order to perform more intelligent selective redo, however, we are using exactly the mechanism of Myers and Kosbie.

9.2. Macro Definition

Our comic-strip timeline visualization is borrowed from Chimera [12]. 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.

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.

9.3. History Enriched Objects

Hill and Hollan [1] describe a set of applications enhanced to show information about the 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 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. Moving a slider time travels back to 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. Since TimeScape does not track changes to the content of documents, it doesn't implement undo or scenario exploration.

Interlocus [15] takes discrete snapshots of workspaces, allowing the user to return to previous appearance states, providing non-selective undo.

Lifestreams [16] generates a visualization of documents organized by time, forming a personal history. Document icons form a stack extending back into the screen based on their timestamp. Documents past or future can be cloned, allowing multiple versions to be created. Subsequences of documents can be grouped and shared, allowing bboards for instance. The system does not keep track of versions or track changes.

Lifelines [17] is also 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 these events can't be selectively undone in our universe, nor can we try out multiple scenarios.

Learning Histories [18] extend the interface of Lifelines to physical simulation programs where undo and modification is possible. In their example for 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. We chose not to maintain timestamps because of the potential for time travel paradoxes. Perhaps because they are traveling in domain time while we travel in user time the problems of self-referential cycles do not come up. They do not support branching histories, and this perhaps also reduces the chance of paradox. Whether all these features can be combined consistently is a very interesting research question.

Meng et al [19] 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, Visage's Time Travel interface intrinsically provides this capability. As in any frame, DQ sliders can be added and set to filter the interface events by type. A visual query linking events to their history objects would allow DQ filtering to display only events having at least one history object mentioning a given object. Events from the TimeTravel interface could even be copied and dragged to a scatterplot organizing them by attribute and object type. Then those events for selected attributes and object types could be dragged to a point on the context tree to effect selective redo.

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, Lifestreams, 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. In addition to the procedural view of histories as sequences of events, declarative representations of exploration scenarios can be used to create information appliances by demonstration.

11. 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. 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.

6. Kolojejchick, J.A., S.F. Roth, and P. Lucas, *Information Appliances and Tools in Visage.* Computer Graphics and Applications, 1997. **17**(4): p. 32-4.

7. Roth, S.F., *et al.*, *Towards an Information Visualization Workspace: Combining Multiple Means of Expression.* Human-Computer Interaction Journal, 1997. **12**(1-2): p. 131-185.

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

9. 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.

10. Guha, R.V., *Contexts: A Formalization and Some Applications,* in *Computer Science*. 1991, PhD Thesis, Stanford: Palo Alto. p. 147.

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. 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.

13. Guha, R.V. and D.B. Lenat, *Cyc: A Mid-Term Report.* AI Magazine, 1990. **11**(3): p. 32-59.

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. 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.

16. Freeman, E. and D. Gelernter, *Lifestreams: A Storage Model for Personal Data.* ACM SIGMOD Bulletin, 1996(March), p. .

17. Plaisant, C. and B. Shneiderman, *An Information Architecture to Support the Visualization of Personal Histories.* Information Processing & Management, 1998. **34**(5): p. 581-597.

18. Plaisant, C., *et al.*, *The Design of History Mechanisms and their Use in Collaborative Educational Simulations*, . 1999, University of Maryland Human Computer Interaction Laboratory. Technical Report 99-11.

19. Meng, C., *et al. Visualizing Histories for Selective Undo and Redo*. in *Third Asian Pacific Computer & Human Interaction*. 1998. Kangawa, Japan: IEEE, p. 459-464.