Proceedings of Intelligent User Interfaces (IUI '01), Santa Fe, NM, January, 2001, pp. 57-64.

Example-based generation of custom data analysis appliances

Mark Derthick
Carnegie Mellon University
Human Computer Interaction Institute
Pittsburgh, PA 15213 USA
mad@cs.cmu.edu

Steven F. Roth
Carnegie Mellon University
Robotics Institute
Pittsburgh, PA 15213 USA
roth@cs.cmu.edu

ABSTRACT

Custom interfaces, which we call *appliances*, allow users to efficiently carry out specialized tasks. Without one, a user is often required to perform repetitive mechanical steps using general purpose interfaces, which we call *tools*. Much research has attempted to enable non-programmers to create appliances for themselves.

We present a system in which a user can choose an example of the task behavior to be automated from a visualization of his past operations. The example is transformed into a visual language, using two simple rules to generalize from the single example to a class of tasks. The user can then edit this representation directly, or continue to refine the example using selective undo and redo. The visual representation can be transformed into an esthetically pleasing appliance by deleting irrelevant components, and rearranging, resizing, and relabeling other components. Restricting the domain to data analysis tasks enables a well-matched visual query language to be used. Appliance interactions are automatically provided by the underlying interactive visualization system in which the appliance is embedded.

An observational study suggests that this system represents a useful point on the ease-of-use vs. expressive power tradeoff appropriate for data analysis, and that the ability to choose and modify examples after the fact is helpful.

Keywords

Programming with Examples, GUI Builder, Visual Query Language.

INTRODUCTION

A study of presentation slide creation showed that the tradeoff between task-specific and generic application software is complex. The authors conclude that providing collections of interoperable tools and appliances may offer the most efficiency and flexibility [1].

Access to custom applications does not require every end user to create appliances. Spreadsheets are a good example of how local experts can create custom models used by a community that shares similar domains and tasks [3]. One reason that non-programmers can create spreadsheet models is that the domain is limited to data manipulation, and models are directly tied to a simple visual representation. Our system similarly capitalizes on domain specificity.

Programming by demonstration (PBD) is an approach to creating custom interfaces without requiring programming expertise. A user tells the system when he is beginning to demonstrate a desired behavior, and when he has finished the demonstration. The computer then attempts to generalize the behavior so that it applies to the whole class of tasks the user may perform in the future. Usually multiple demonstrations are required to clarify which aspects of the behavior are fixed, which should be parameterized, and the degree of generality of the parameterization. In practice, finding a set of examples that spans the task space requires sophisticated abstract reasoning on the user's part, and can be quite frustrating. As a result, programming by demonstration has not yet been widely successful. The broader class of attempts to generate interfaces automatically using heuristics has suffered from a similar unpredictability. The relationship between the specification and the final result can be hard to understand and control [4].

We present an alternate approach, in which a single example is transformed into a declarative visual representation of the structural relations among its operations. This structure is intended to be more comprehensive than the user's task requires, so editing it consists of deleting unneeded components. This explicit editing takes the place of heuristics. We believe that non-programmers can do this deleting, even if they cannot construct such a representation. As with spreadsheets, we rely on the fact that the domain of interest (data manipulation and visualization) is quite limited to provide a reasonably simple visual query language representation.

The next section presents an example exploration scenario that a user would like to automate. Sections 3 and 5 describe the visual query language and the appliance editing process. Section 4 presents the algorithm for inferring the query. Section 6 discusses the results of an informal user evaluation. Related work is discussed last.

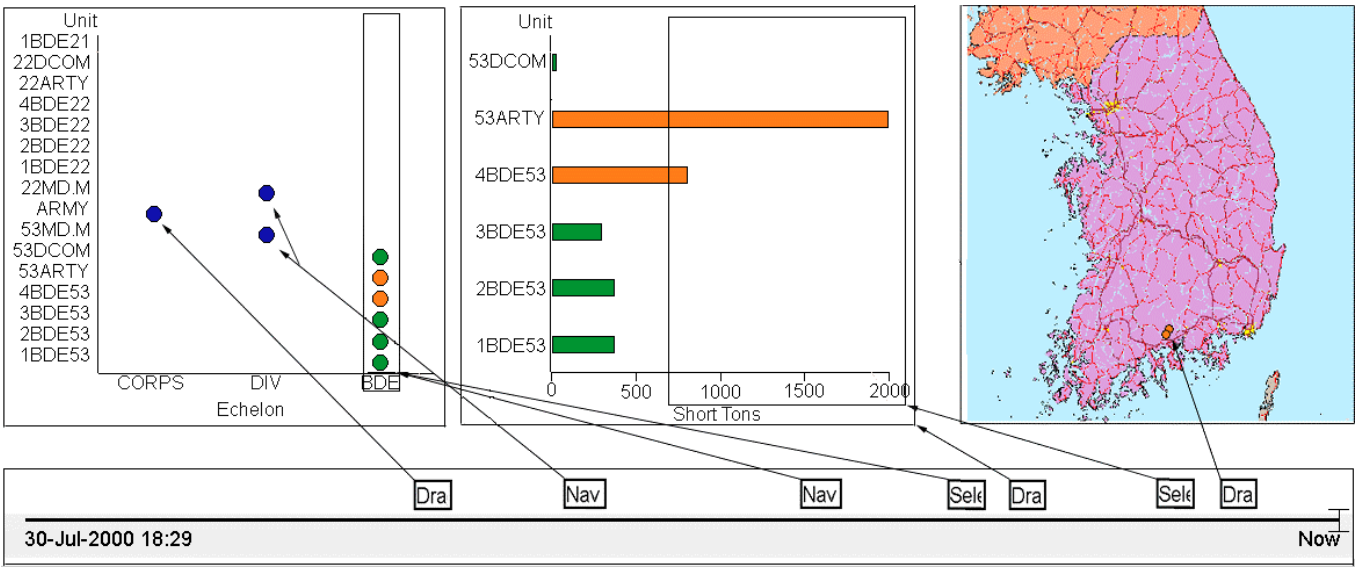

Figure 1 The timeline of user operations is shown at bottom. Arrows were added in Photoshop to link each operation to its result set on the three visualizations above. The timeline represents a span of about a minute.

EXAMPLE ANALYSIS SCENARIO

Our system is built on top of the Visage data exploration and visualization system, developed by Maya Design Group and Carnegie Mellon University [5]. The following example was used in the video accompanying our IUI'00 paper (http://www.cs.cmu.edu/~sage/animations/IUI.rm). That paper examined how reified representations of user actions supported browsing, comparison of multiple scenarios, and selective undo and redo [6]. This paper shows that the same representations can be used as input to a system that generates custom interfaces.

The user is a transportation analyst whose job is to ensure adequate supplies to each brigade of an Army corps. He must evaluate the amount of supplies needed and access to supply routes. For readers unfamiliar with military terminology, the point of the example is that a sequence of operations is performed, and that a causal ordering is imposed when the output of earlier operations serves as the input for later operations.

In Figure 1, the analyst has created three visualizations using the expert system SAGE and its sketch-based user interface, SageBrush [7]. He then **Drag**s a graphical representation of the corps to the echelon chart on the left[1].

From the corps unit, he **Navigate**s down the `subord_unit` relation to find two divisions, and in turn **Navigate**s from the 53rd division to its six brigades. Each of these actions adds an event to the timeline interface (wide bottom rectangle). He **Select**s only the brigades, using a bounding box to paint them green.[2] Note that the bounding box extends above and below the y-axis. The event is therefore heuristically annotated with the inferred intention of selecting any unit that is a brigade.

A copy of the brigades is **Drag**ged to a bar chart showing their `total_supply_weight`. Those with high needs are **Select**ed, this time in orange. Again the bounding box extends above and below the y-axis, as well as to the right of the x-axis, and Visage infers an intention to select any unit whose `total_supply_weight` exceeds 690 tons. Copies of the two high supply-need units are **Drag**ged to a map, where units far from ports or highways would then be examined in more detail.

Each time the operational plans change, the transportation analyst must repeat this task. The next section discusses the visual representation used to generalize the data manipulation effected by the seven operations in Figure 1.

VISUAL QUERY ENVIRONMENT REPRESENTATION

VQE combines the expressive power of database query languages with the interactivity of direct manipulation data visualization systems [8]. It was originally designed as a visual interface for constructing queries manually. It uses the familiar node-link diagram to express database joins, which correspond to **Navigate** operations in Visage. It uses Dynamic Query [9] sliders and bars to represent database range and equality restrictions. These correspond to **Select**ion of graphical objects (which we call *graphemes*) individually or with bounding boxes in Visage.

[1] High resolution color figures are available at, e.g., http://www.cs.cmu.edu/~sage/papers/IUI01/Fig1.GIF.

[2] Visage uses a form of selection called "brushing" [2], which allows coordination across visualizations. The two orange units in the echelon chart were green until the selection in the bar chart changed them.

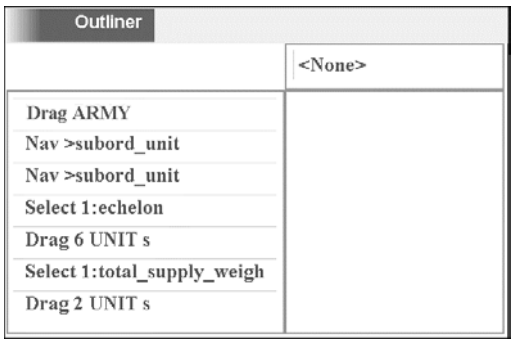

Figure 2 Full names of the operations in Figure 1.

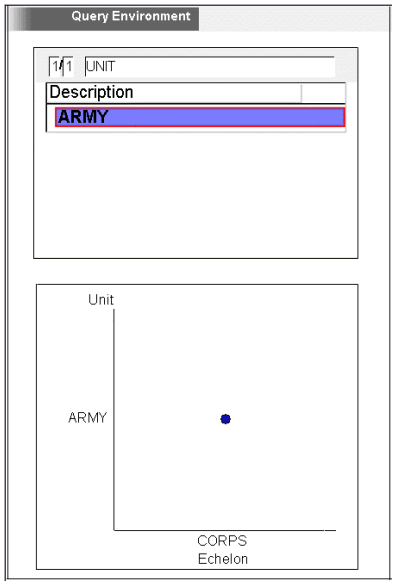

Figure 3 VQE after the first **Drag** operation.

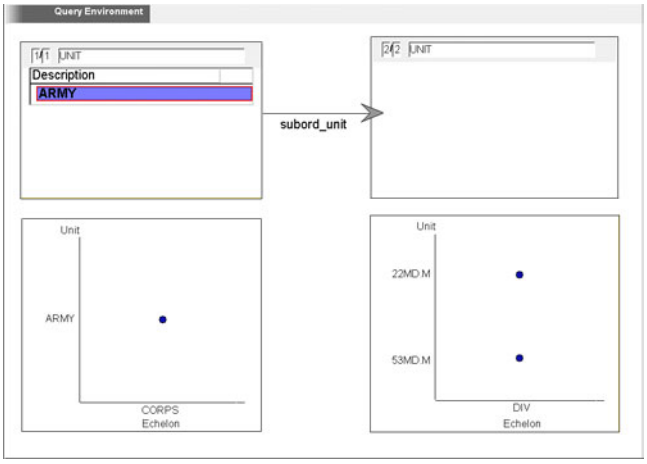

Figure 4 VQE after the first Navigate operation.

In normal usage, a user would select all the operations in the timeline of Figure 1 and drag them to a VQE as a group. For expository purposes, we illustrate the cumulative effect on the query of dragging each of the operations into VQE in chronological order. Figure 2 shows their full names.

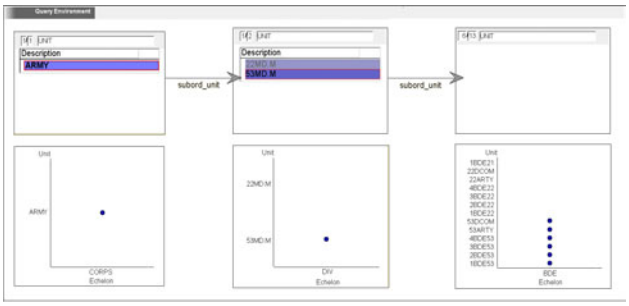

Figure 5 VQE after the second Navigate operation.

Figure 3 shows the VQE state after the Army corps has been **Drag**ged into the echelon chart. The single node query graph (upper enclosed rectangle) shows that the query involves a single Army unit. The generalization algorithm does not know why this unit was chosen for dragging, so picks it out from other candidates using the default attribute `description`, requiring it to have the value ARMY. VQE also shows all the visualizations in which this set of units was manipulated. Here the corps unit only appears in the echelon chart. Visualizations stacked below a query node show the data objects in that node.

Figure 4 shows the state after the first **Navigate** operation, from the corps unit to its two divisions. The query graph now consists of two nodes linked by the `subord_unit` relation. Each of the nodes represents a set of units, but the ones on the right are subordinates of the ones on the left. The divisions also appear only in the echelon chart. Note that there are two copies of the echelon chart, one for each set of units.

Figure 5 shows the state after the second **Navigate** operation. Since VQE does not know why the 53rd Division was chosen as the source of the navigation, it adds a `description` DQ widget with only the 53rd selected. The query node header now expresses that only one of its two UNITs satisfies the current query constraints. As a result, its 6 brigades are visible, while the 7 brigades of the 22nd Division are invisible.

Figure 6 shows part of VQE after **Drag**ging the 6 visible brigades to the bar chart, and **Select**ing those with high `total_supply_weight`. Only two units satisfy the conjunction of the two DQ constraints, and are visible in both the `echelon` chart and the `total_supply_weight` chart. The top DQ represents the [vacuous for this dataset] constraint that `echelon` = Brigade. The bottom DQ represents the range restriction that `total_supply_weight` > 690 Short Tons.

Figure 7 shows the result of **Drag**ging the two visible units to the map. The only change is the additional visualization.

Now the analyst cleans up VQE to make a user-friendly appliance. Figure 8 shows the state when he is almost done. He has dragged the `total_supply_weight`

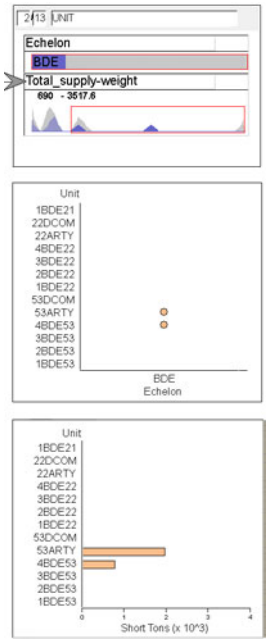

Figure 6 The third column of VQE after the second Select operation.

Figure 7 VQE after the final **Drag** operation.

slider out of its query node onto the appliance background. He dragged the two `description` DQ widgets to the trash, because he does not want to limit the corps or divisions that can be examined. He leaves the `echelon` widget inside its query node when he iconizes all the query nodes, because the end user will never need to change this value. He has edited the text describing the DQ slider, and the appliance as a whole (see Figure 11 for legible text). He has opened the tool drawer (gray column on left), which contains all the controls necessary to construct queries manually. Next he will uncheck the box indicating

Figure 8 Partial layout customization.

whether iconized nested frames should be visible. Then he will close the drawer and resize the window. The appliance is then ready for use. The analyst can change the corps he is checking by dragging into and out of the echelon chart. He can see the supply needs of each brigade by adjusting the slider until it becomes visible. The appliance works in coordination with other Visage tools and appliances through drag and drop and brushing.

Because the program representation is based on declarative queries, the order of operations is not restricted to that of the example. For instance, by dropping a single brigade on the map, VQE will look up its division and corps, and in turn all their subordinate brigades, filter them, and display all the problem brigades on the map.

QUERY CONSTRUCTION ALGORITHM

Query construction is algorithmic. While using bounding box bounds to "infer" range restrictions may be considered a heuristic, we prefer to consider it an extension to the semantics of the selection operation. We expect users to consciously take advantage of it in performing operations. Similarly, part of the semantics of other operations is that objects that are operated on together are expected to play the same role in the appliance. They constitute the query nodes in the construction interface. Their relations to other nodes and their DQ settings define their meaning. By this rule, the brigade query node aggregate contains all 13 brigades of both divisions, even though only one division was involved in the example. The explicit choice of the 53rd division is separately represented with DQ. These two semantic extensions constitute the only built-in generalization of examples.

The query graph is the central organizing feature of both the construction algorithm and its user interface. The query graph is displayed linearly, chronologically ordered by the first operation performed on each node's aggregate. Visualizations that served as a source or destination of any operation on this aggregate are stacked below the corresponding aggregate node.

When a new operation is dropped in VQE, it is first recursively expanded into all its lowest-level subevents.

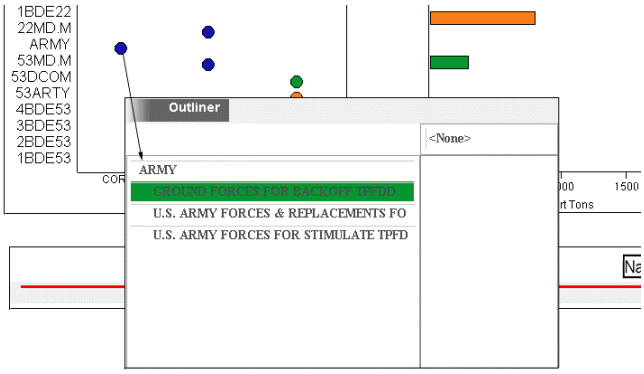

Figure 9 Performing additional operations to modify the appliance. The arrow indicates dragging a copy.

Those that are one of the three basic operations (**Select, Navigate, Drag**) are then processed individually.

First, the operation's origin and/or destination aggregates are determined. **Select** and **Drag** have only a destination aggregate; **Navigate** operations have both. If no match exists in the query graph, a new node is added. If the operation is a **Navigate,** the origin and destination nodes are linked by the relation navigated across.

Determining whether an operation's aggregate matches that of an existing node requires examination of Visage's internal representations that are not normally seen by users. In addition to graphemes denoting domain objects, they are also denoted by data objects called "implementation objects." An operation's aggregate is declared to match that of a query node if there is an intersection of the implementation objects each maintains. Requiring only intersection rather than equality allows the destination of the first navigation operation in our example to match the origin of the second, even though the latter uses only one of the two divisions to navigate from. Requiring intersection of implementation object rather than domain objects ensures that the aggregates share the same role in the chain of user operations.

If the operation is a **Select**, Dynamic Queries are then added for the appropriate attribute(s). Finally any visualizations containing the origin or destination graphemes are added below the respective query nodes, unless already present.

MODIFYING THE INTERFACE

Imagine that our analyst is assigned to support a particular operation plan involving an ad hoc group of army units (called a Force Module). He would like to continue analyzing one corps at a time, but would like to select it from a list of units currently in the force module. Second, his task is to take care of the logistical needs of only those brigades whose `mission` is DEFEND.

Now his goal is to make these specific changes to the appliance, rather than to explore additional data. Therefore he does not bother to create custom visualizations to show the additional attributes. He adds an Outliner to the Visage desktop, a tool that supports navigation to objects of any data type and display of any attribute. He copy-drags the corps unit to the Outliner and navigates back up the `component` relation to the three force modules it belongs to. He then selects the relevant force module (Figure 9).

Dropping the new **Drag** and **Navigate** operations on the appliance completes his first change (see Figure 10). Note that the rule to order the query nodes chronologically has placed the Force Module node on the right, resulting in line crossings. A better layout choice might be the conventional approach of minimizing line crossings.

Rather than modifying the example further, he chooses to edit the interface representation directly. The drawer containing the DQ and other tools has been pulled out using the mouse. It is normally open during query construction and closed once the application is complete. He drops in a DQ tool from the tool drawer onto the

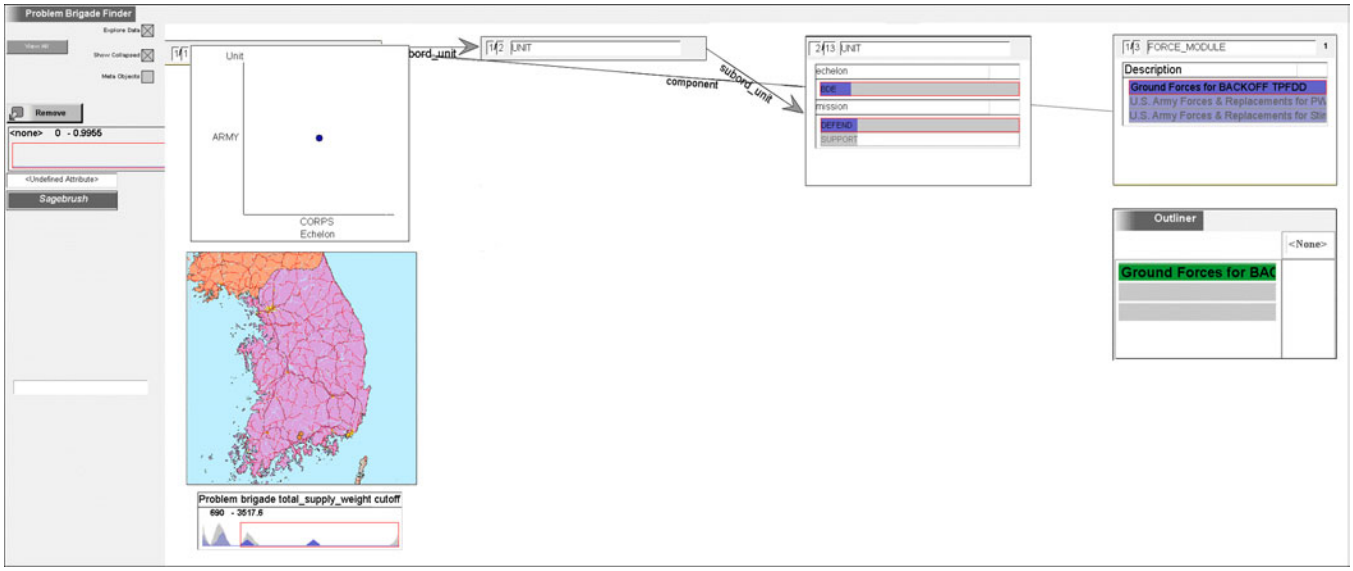

Figure 10 VQE after adding additional operations.

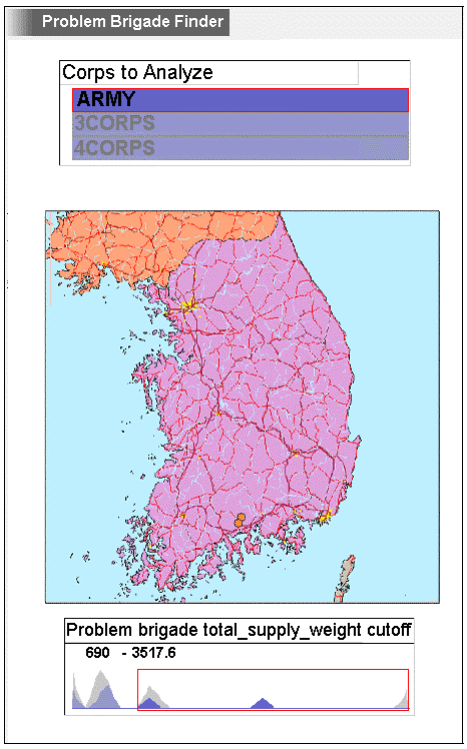

Figure 11 Final appliance appearance.

brigade query node, chooses the `mission` attribute, and activates only the value DEFEND. This action is also reflected in Figure 10.

Finally, he lays out the appliance using normal GUI builder operations. In Figure 11, he has dragged the `description` DQ widget out of the corps query node to the top of the appliance and relabeled it. This widget had been deleted in Figure 8, but the new **Drag** operation restored it. He has deleted the echelon chart because he will now select the corps using DQ rather than drag and drop. The Outliner has also been deleted. To use the appliance, a corps is selected from the list provided by the Dynamic Query widget. Then the DQ slider for `total_supply_weight` is adjusted to pick out the problem brigades. These can be dragged to other Visage tools for more detailed examination.

INFORMAL EVALUATION

Our system supports open-ended user behavior involving a combination of visualizations, direct manipulation operations and their graphical representation, and the query graph representation. Further, there is no other system that we can compare with fairly, because we take advantage of the limited domain of data exploration. Under these conditions, focused questions answered by controlled statistical studies of a large number of users are rarely valuable. They often fail to generalize to an interesting class of real-world situations, only a few variables can be manipulated, and even statistically significant results are often not large enough to be important [10, p 148].

Therefore we chose to perform an observational study. Detailed studies of users working with a system can provide valuable insight even if performed with few subjects [10, p 136]. We watched subjects as they performed a task, and asked them to "think aloud" as they worked.

Two subjects were asked to construct an appliance for performing the analyst's task as described above, with the additional requirement that the divisions be selectable from a menu. Each was proficient in using Visage (and moreover was a developer), but had never used VQE (although Subject B was familiar with it conceptually). Each read an introductory paragraph about the appliance generator, and the experimenter demonstrated an example of its use. This example involved adjusting the scheduled arrival times of Army units based on their roles in the operation.

The task description explained that the user must examine groups of brigades for supply problems, indicated by `total_supply_weight` and map location. There should be one group for every division, and each division subordinate to the user's corps should be selectable. The three visualizations and timeline slider in Figure 1 were given to the subjects initially (with no data in them).

Both subjects were able to create an appliance in 15-20 minutes. Both solutions resembled cleaned up versions of Figure 8, although the operations used to generate the example differed significantly from those given above. They preferred to use Visage's Outliner (drill-down table) to do most navigation, and for selection of discrete attributes. Then the results were dragged to the bar chart and map just to select the range of `total_supply_weight`, and to look for the problem units.

Due to an unclear task specification, Subject A's appliance required dropping divisions into the appliance, rather than selecting from a menu. A was asked to modify his appliance to make this possible. Subject B was asked to address the modified task described in the previous section. Each subject completed the modification within 5 minutes.

Each subject completed a survey of how he preferred to use the interface, difficulty understanding and editing the operation and VQE representations, and overall satisfaction. Our hypothesis was that the ability to iteratively improve the example would be an advantage over systems that require a user to press record, execute an example perfectly, and then press stop. This hypothesis was generally supported. Subject B generated an almost optimal sequence of operations on the first try. He spent about 30 seconds examining his operations in an Outliner and eliminated two extraneous operations, with the result that when he dropped the remaining operations on VQE he got exactly Figure 7, except with Outliners instead of the echelon chart. Thus it was only when he was given the

modified task that going back and fixing the example was an issue. He used the original Outliner to perform several new operations, but attempted to do extra work to make these operations connect with the old ones. Both he and VQE became confused. He attempted to start over with a fresh VQE and a fresh set of new operations (just the single drag, navigation, and select this time). But data structures had been damaged and the experiment was halted.

Subject A generated many more operations in his example. He was able to screen out many extraneous operations based on their order and text description in an Outliner. He also used the timeline interface to return to the state before and after several operations in order to disambiguate which reified operation stood for which real operation. However the operations he dragged to VQE formed a rather baroque logical progression and he was confused by the structure displayed in VQE. He started over with a new VQE, using the timeline slider to undo all his actions. The second time the route was less baroque, but still far from optimal. He repeated some operations to get a clean set that he expected to work together, even though they were not recorded in causal order. Except for the fact that he did not realize the need to perform two subord_unit navigations, the resulting structure was correct. He easily went back to the original Outliner and performed an additional navigation operation, which correctly transformed the VQE structure.

Both users easily and quickly accomplished the aesthetic cleaning by deleting unneeded visualizations, hiding query nodes, and relabeling frames. Thus "programming by deleting" was seen to be effective even though they did not know how to use VQE constructively. (This process had not been described to the subjects.)

Both users felt strongly that it was hard to associate the reified operations with the real operations. Some suggestions emerged: 1) The ability to step through operations individually and choose whether each should be part of the example. 2) An extension of brushing so that selecting a reified operation would also color the frames and graphemes involved in it. 3) Showing before/after screenshot pairs of the affected areas. More research is required in this area.

Correspondence between the VQE declarative representation and the original operations was unclear to the subjects at the detailed level, but that was not of concern to them. The way that it portrayed the dynamic operations in a declarative high level view was clear for B and "really cool" for A, who had "never thought about the structure of his exploration that way."

Both users naturally went back to their original examples in order to modify them. There was little effort required to start with a fresh VQE, and so this was a second choice. Starting the example over was a clear third. B said "now that I know my operations are being recorded, it seems like a total waste to have to do them over again."

Both strongly agreed that this was a much better way to generate appliances than by writing code, and felt that the expressive power is not a major problem. They still envision tweaking the result by writing code, for instance substituting radio button behavior for the checkbox behavior of the DQ bar chart widget. A said "it looks like it's almost ready to do real work." The qualification refers to the robustness of the implementation as well as the operation-matching problem. Both strongly preferred using this tool to repeating the operations by hand, and both preferred it to unadorned VQE (though neither had in fact used VQE to build anything).

After addressing the problems identified above, the system should be tested with non-programmer Visage users.

RELATED WORK

Query by Example (QBE) [11] is a database interface where users place variables in a visual representation of the database schema, rather than using a textual language like SQL. VQE is somewhat similar in its visual language. It differs in that it gives continuous feedback based on the set of actual data that has been dropped on it. This paper goes a step further toward real examples in that the visual query representation can be generated automatically from operations carried out in Visage's normal interface.

Database form generators allow the user to create applications with charts, tables, buttons, and checkboxes. The data manipulation is specified using a query language. At best this will be QBE. The direct manipulation layout of the components is similar to that in VQE.

Chimera [12] 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. Because it uses procedural macros and shows screenshots of each operation result, Chimera does not have the operation-matching problem that we do. We plan to adopt comic strips for our timeline interface. Chimera does not allow modifying the example after the fact, and it is not designed to work well in the data analysis domain.

Programming by demonstration in general was discussed in the Introduction. Pursuit [13] is an example of PBD that we have tried to emulate in certain respects. Its techniques for enabling the user to understand the visual program include: 1) Programs are specified by operations on real data. 2) Programs are represented in a visual language in which the data and operations of a program look very much like the actual objects and changes users see on the desktop

when constructing the program. The language is similar to Chimera's, except that the images are more abstract than screenshots. 3) Programs appear incrementally as the user executes each operation. Pursuit also emphasizes manipulation of sets of objects, to reduce the need for looping constructs.

The main difference between Pursuit and our system is that we allow the user to retrospectively choose operations to automate, rather than include each operation as it is performed. Our experience is that extraneous operations are confusing in the visual programming representation. The user can still see the effects of individual operations on the program by dropping them individually on VQE. A second difference is the domain. Pursuit still relies on looping and branching constructs for doing file operations in a visual Unix shell. We have avoided procedural constructs altogether by using a declarative query language, which simplifies programming but does not generalize to domains beyond data analysis. On the other hand, no previous PBD system can generate interactive data analysis appliances.

SUMMARY

We described a tool for automatic appliance construction from an example. Using Visage's first class representation of user operations *enables the example to be created piecemeal*, rather than executed perfectly in record mode as required by PBD systems. The chosen operations are dropped into VQE, which shows a visual representation of the causal structure and the relevant visualizations.

The construction algorithm uses two cases of semantic overloading: bounding box selection now has an intentional meaning (the bounds) as well as an extensional one (the graphemes). Second, sets of data objects represented by graphemes that are operated on together are considered to have an intention as well, which comes to be defined by its place in the query graph.

The example can be edited with normal Visage operations, including selective undo/redo commands. Additional operations can even be added after the application is in use. This gives users the freedom to ensure that all constraints required to build the application are present in the VQE representation, so *the only editing that must be done in this more abstract representation is deleting or hiding extraneous information, and visual rearrangement and labeling*. As construction and editing proceeds, the user has immediate feedback about the effect on the visualizations as well as the query structure. In addition, he can test the application at any time by copying the VQE frame and dropping new data on it.

The appliance is integrated into Visage's full-featured data exploration environment, so no new widgets or behaviors must be programmed. The domain-specific design trades off expressive power and ease of use in a new way. The use of a declarative query language designed strictly for data exploration avoids the need to express procedural constructs like branching and looping, which have been difficult to infer in previous PBD research.

Informal testing suggests that being able to refine an example based on feedback from the resulting appliance is extremely valuable. Users preferred this to starting over, or refining using VQE techniques. *Subjects felt that most appliances they currently build can be reasonably well approximated using this new tool, and that it is far easier to do so*. If even developers feel they can build appliances this way, surely the target users will find its expressive power more than adequate. However we must simplify the task of recognizing reified operations.

REFERENCES

1. Johnson, J.A. and B.A. Nardi, *Creating Presentation Slides: a study of user preferences for task-specific versus generic application software.* ACM Transactions on Computer-Human Interaction, 1996. **3**(1): p. 38-65.
2. Becker, R.A. and W.S. Cleveland, *Brushing Scatterplots.* Technometrics, 1987. **29**(2): p. 127-142.
3. Nardi, B.A. and J.R. Miller. *An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development*. in *Proceedings of the Conference on Human Factors in Computer Systems (CHI'90)*. 1990: ACM Press: p. 197-208.
4. Myers, B., S.E. Hudson, and R. Pausch, *Past, Present, and Future of User Interface Software Tools.* ACM Transactions on Computer-Human Interaction, 2000. **7**(1): p. 3-28.
5. 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.
6. Derthick, M. and S.F. Roth. *Data Exploration across Temporal Contexts*. in *Proceedings of Intelligent User Interfaces (IUI '00)*. 2000. New Orleans, LA: p. 60-67.
7. Roth, S.F., J. Kolojejchick, J. Mattis, and J. Goldstein. *Interactive Graphic Design Using Automatic Presentation Knowledge*. in *Human Factors in Computing Systems (SIGCHI)*. 1994. Boston, MA: ACM Press: p. 112-117.
8. 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.
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. Baecker, R. and W.A.S. Buxton, eds. *Readings in Human-Computer Interaction*. Morgan Kaufmann Readings Series, ed. M.B. Morgan. 1987, Morgan Kaufmann: Los Altos, CA. 738.

11.Zloof, M.M., *QBE: A Language for Office and Business Automation.* IEEE Computer, 1981. **14**(5): p. 13-22.

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. Modugno, F. and B.A. Myers, *Visual Programming in a Visual Shell* Journal of Visual Languages and Computing, 1997. **8**(5/6): p. 491-522.