Interface Support for Set-Based Problem Solving

Michael Terry
Everyday Computing Lab, GVU Center
College of Computing, Georgia Tech
Atlanta, GA
mterry@cc.gatech.edu

ABSTRACT
Solving ill-defined problems, such as those found in design, often requires active exploration of multiple, potential solutions. However, few interface mechanisms exist to support this practice. This paper argues for the notion of a *set-based interface*, or an interface that facilitates the generation, manipulation, evaluation, and management of sets of potential solutions. Drawing upon studies of actual design practices, I motivate the need for these support structures, then present three tools I have built to explore these concepts. I conclude by describing a study designed to examine these tools' effects on the problem solving process.

Categories and Subject Descriptors: H.5.2 [**User Interfaces**]: User Interfaces – Interaction Styles

Additional Keywords and Phrases: Exploration, experimentation, what-if tools, interaction models, parallel exploration, previews

INTRODUCTION

For many problems, a single "correct" solution does not exist. Instead, a variety of solutions can be developed, each possessing its own unique set of strengths and weaknesses. For example, there is no single, best solution for the design of a building, software system, or consumer product: new designs can always be created, and old ones improved [6]. These types of problems are often referred to as *ill-defined* problems [6], and span the range from design problems to relatively common tasks such as writing.

Because ill-defined problems can be solved any number of ways, multiple potential solutions are often developed, either as part of an explicit process, or out of necessity. For example, early in the problem solving process, experienced practitioners often intentionally create several rough ideas to better understand the problem and the client's needs [5]. Later, as ideas are instantiated, alternatives are also explored on an ad-hoc basis when it is unclear how to best implement individual elements of the solution [8].

UIST '04, October 24–27, 2004, Santa Fe, New Mexico, USA
ACM 1-58113-962-4/04/0010

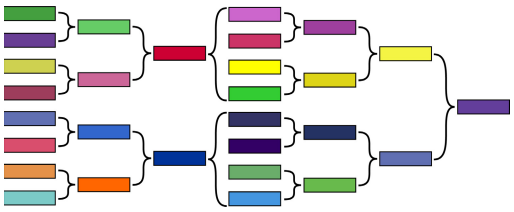

Figure 1. Set-based problem solving: Multiple alternatives are continually generated, evaluated, and pruned

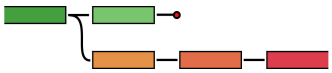

Figure 2. Point-based problem solving: A single solution is revised until acceptable. Dead-ends (the top-right red dot) require backtracking

A series of studies examining the design process at Toyota [7,12], as well as my own study of expert users of an image manipulation application [8], indicate that active exploration of alternatives is a common problem solving strategy. These studies further suggest that this practice often extends beyond initial exploratory design phases, and well into later implementation phases.

In the study of Toyota's engineering practices, the degree to which engineers prototype and explore alternatives led Ward *et al* [12] to describe the practice as "set-based concurrent engineering," since sets of solutions are repeatedly generated, evaluated, and pruned (Figure 1). In this paper, I focus on individuals, rather than teams of designers, and use the phrase *set-based problem solving* to describe this general problem solving strategy.

Set-based problem solving can be contrasted with *point-based problem solving* [7,12], a methodology marked by continual revision of a *single* solution instance. While a set of potential solutions may initially be considered, only one is ultimately chosen before detailed design commences (the goal "point"). This chosen solution instance is then revised until it reaches an acceptable end state (Figure 2). As should be obvious, set- and point-based methods are not

mutually exclusive. Nonetheless, they do represent distinct ideologies regarding how to best solve a problem.

While my study and others [5,7,12] indicate that exploring solution variations is a vital and desired strategy, there exists little explicit support for this practice in current user interfaces. Instead, interfaces tend to focus users' attention on one solution at a time, promoting a point-based style of problem solving: a single solution instance (e.g., a document) is continually revised until complete. With little support available for working with sets of possibilities, users must develop their own ad-hoc procedures to manually generate, manipulate, and evaluate alternatives. As a simple example, when choosing a typeface for a document, an individual may find several fonts promising. However, users typically cannot directly compare these alternatives in the context of their document. Instead, they need to engage in practices such as repeatedly trying and undoing the different fonts in order to evaluate them. The number of steps required to perform these comparisons can ultimately discourage exploration, and potentially affect the quality of the final solution.

This work seeks to address these interface shortcomings so that individuals can more easily engage in set-based problem solving practices. Accordingly, it seeks to make the following contributions:

- establishment of a basic set of interface design principles to support set-based practices
- development of a set of tools illustrating these concepts
- an assessment of how these tools impact the problem solving process and the quality of the solutions that result

In the rest of this paper, I first review related efforts in the HCI literature, indicating ways this work can be extended. I then restate my specific research problem and goal. Next, I introduce the notion of a *set-based interface*, and present three example systems I have built exploring this concept. Finally, I consider the challenges and open issues surrounding this work, and describe a study aimed at assessing the impact these types of tools have on the problem solving process.

BACKGROUND

A number of interface mechanisms have been developed to support the process of exploring alternative solutions. In general, these efforts can be grouped into three categories:

1. History tracking tools
2. "What-if" tools
3. Enhanced previewing mechanisms

Automated history tracking tools, such as Timewarp [2], more completely capture a user's history by recording and representing all states visited in a branching tree structure. Other history-based tools take a more conservative approach, and simply increase the ease with which users can store and access snapshots of important states [11]. In all cases, history tracking tools support exploration by reducing the work associated with saving and accessing notable states.

What-if tools, on the other hand, enable users to more easily explore alternative possibilities in parallel. The common spreadsheet is perhaps the best example: it provides the structure to embed alternatives within the same document, and the tools to facilitate comparisons between them. For example, several alternatives can be placed side-by-side in separate columns, then evaluated using a chart or graph. This same grid-based organizational scheme has been applied to other domains, such as scientific visualization [1], to achieve similar effects.

Other what-if tools instantiate the core concepts in a different manner. Aran Lunzer's subjunctive interface [3] allows objects to exist in multiple states at the same time. For example, a canon in a physics simulator can be set to multiple angles, then "fired" to view the trajectory of a cannonball for each angle specified. This capability allows users to try several possibilities simultaneously, without needing to create separate solution instances.

While history and what-if tools provide explicit structures for holding alternatives, the creation of these alternatives is, by-and-large, manual. Enhanced previewing mechanisms take a more proactive stance and automatically generate sets of potential future states for a user to consider. Design Galleries [4] automatically generates hundreds of possible solutions, then filters these results to present only the most semantically distinct alternatives. For example, in lighting a 3D scene, Design Galleries will systematically vary the number, location, and intensity of lights, then present users with the most visually distinct results. Users then choose the most desirable result to proceed.

Additional Opportunities for Supporting the Exploration of Alternatives

The categories of tools described above provide a valuable starting point for better supporting set-based problem solving practices. However, there are a number of ways these concepts can be improved upon. These improvements focus on methods of producing and manipulating alternatives, and on tool integration.

In general, few mechanisms exist that automatically generate *sets* of alternatives for the user to consider. Both history tracking and what-if tools provide support for holding alternatives, but offer few capabilities for actively generating alternatives. Enhanced previewing mechanisms generate sets of possibilities, but these sets are transient: Users cannot choose multiple points of interest, but must instead commit to just one future state at a time.

To better support set-based practices, interfaces should, when and where possible, generate sets of alternatives for

users to consider, and provide capabilities for selecting *several* of the possibilities for further exploration. As a simple example, systems such as Design Galleries could provide capabilities whereby users select several of the resultant states to place directly within a what-if or history tracking tool.

Once generated, alternative possibilities often require further iteration. To facilitate working with sets of possibilities, users should be able to treat solution instances as first-class user interface objects. For example, users should be able to apply an operation just as easily to a set of solution instances as to one. These types of capabilities are particularly useful when sets of alternatives are more similar than they are different, implying similar changes need to be made to each.

Finally, set-based problem solving practices can best be supported when these various mechanisms are integrated. Enhanced history mechanisms, what-if tools, and previewing mechanisms all work well on their own, but their individual features complement one another.

SET-BASED INTERFACES

To summarize the research problem, I seek to support a style of problem solving in which an individual repeatedly produces, compares, and prunes sets of possibilities. Accordingly, I propose the concept of a *set-based interface*, or an interface that facilitates generating, manipulating, evaluating, and managing sets of alternative solution instances.

To explore this concept, I have created three tools: Side Views [9], Parallel Pies [10], and *partials*. These interface mechanisms operate in the domains of word processing and image manipulation; image manipulation; and programming, respectively. I briefly describe each of these tools in turn.

Side Views provide on-demand, persistent previews of one or more commands and their parameters. Initially, previews appear in the context of a tool-tip window. However, Side Views can be made to persist by clicking on them. Multiple Side View windows can be instantiated simultaneously, affording side-by-side comparisons of commands. Side Views can also show ranges of previews for each command parameter, rather than just one preview for the entire command (Figure 3).

While Side Views offer users capabilities to quickly generate and evaluate sets of transient possibilities, Parallel Pies provides the structure and facilities to work with a more permanent set of alternatives, over a longer period of time. Akin to a what-if tool, Parallel Pies allows multiple solution instances to be added to the same workspace. A visualization tool evenly divides the workspace to show each alternative in its own "slice" (Figure 4). Users can selectively view or hide each variation by toggling a thumbnail representing the alternative.

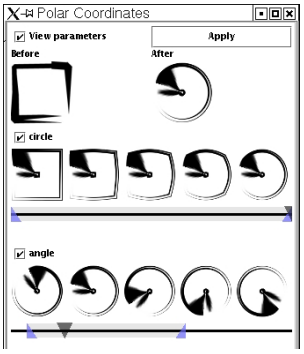

Figure 3. Side Views automatically generate previews of multiple, potential, future states, as in this example for a polar coordinates filter. For each parameter, an adjustable range of values is previewed via five images

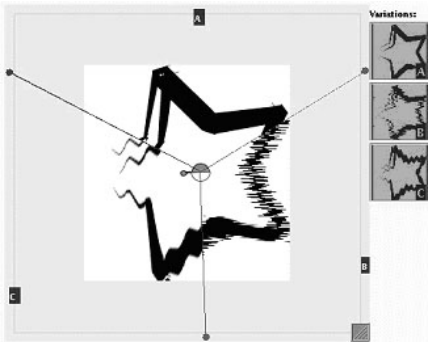

Figure 4. Parallel Pies support direct embedding of multiple alternatives in the same workspace. Three different solutions are shown in the above workspace, each with its own visualization slice and corresponding thumbnail

New solution instances can be added to the workspace via an "Add Variation" operation available within command dialog boxes. When invoked, Parallel Pies automatically duplicates the current document state, applies the given command to the copy, then inserts that new solution instance into the workspace. The original state of the document is kept intact, visible in its own slice, with the new solution instance assigned its own slice. This capability allows users to quickly add any and all interesting variations as they are encountered. Finally, Parallel Pies allows multiple solution instances to be simultaneously manipulated and operated upon, as if they were a single entity.

While the previous tools are well suited to visual tasks, *partials* applies the principles of a set-based interface to the domain of programming. A *partial* is an extension to the Java programming language that allows a programmer to express a set of possible values for a given expression. For example, *partial(int : 1, 8, 11)* represents an expression that can assume the integer value of 1, 8, or 11. At runtime, when this expression is reached, the programmer is presented with a list of choices to choose from (1, 8, or 11 in this case), plus the opportunity to input a custom value.

Using this mechanism, a programmer can explicitly state the possibility for multiple alternatives within the source code itself, and test the different possibilities each time the expression is evaluated at runtime. Partials offer a well-defined mechanism for describing alternative possibilities in a programming environment, replacing ad-hoc methodologies to test alternative scenarios, such as conditional compiles.

FUTURE DIRECTIONS

Supporting set-based practices creates additional challenges for user interface designers since they must consider how users may want to explore alternatives in the course of performing their work. Since few interface design examples exist, more work is necessary to understand how these concepts are best instantiated in different task domains.

While some tool examples have been built, few studies have assessed the effect they have on work practices. A recent study by Lunzer and Hornbaek [3] indicates that an increase in task performance is possible when using these tools for solving well-defined problems, but more studies are necessary to understand their effect when solving ill-defined problems.

Given these needs, this research has two near-term goals. The first is to better map out the design space for set-based interfaces. In particular, my existing work has primarily investigated *data manipulation* tasks, in which a known set of data is repeatedly transformed into a more desirable state (e.g., image manipulation). An open question is how these principles transfer to *content production* tasks, such as writing. I would thus like to develop additional tool designs for other task domains to address this question.

The second goal is to conduct a study assessing these tools' effects on the problem solving process. To this end, I plan on conducting a controlled study that compares user performance and problem solving strategies with and without the aid of these tools. In particular, I will test the impact of Side Views and Parallel Pies in the task of developing a product logo. In this study, subjects will be given the ill-defined task of colorizing a logo to jibe with the spirit of a marketing campaign. Individuals will be randomly assigned to one of two conditions in which the research tools are, or are not, available.

This study will attempt to shed light on the following questions: Given tools designed to support set-based practices, do individuals actively make use of these capabilities? That is, do subjects generate and pursue more alternatives in greater depth, or does their problem solving strategy remain relatively unchanged? Is there a significant difference in the quality of solutions produced between the two conditions? Are users led off task when presented with multiple possibilities? That is, do they "wander" more at the cost of making forward progress? Is there any difference in the cognitive load imposed on users when using these tools?

The results of these final two pieces of my research should serve to direct and focus future efforts geared towards supporting set-based problem solving practices.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Beth Mynatt, for her continual support and guidance. This work has been funded via a grant from the Sloan Foundation.

REFERENCES

1. Chi, E.H., Riedl, J., Barry, P., and Konstan, J. Principles for information visualization spreadsheets. In *IEEE Computer Graphics & Applications*, vol. 18, no. 4, July-August 1998, 30-38.
2. Edwards, W.K., and Mynatt, E.D. Timewarp: Techniques for autonomous collaboration. In *Proceedings of CHI 1997*, 218-225.
3. Lunzer, A., and Hornbaek, K. Usability studies on a visualization for parallel display and control of alternative scenarios. In *Proceedings of AVI 2004*, 125-132.
4. Marks, J., *et al*. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of SIGGRAPH 1997*, 389-400.
5. Newman, M., & Landay, J. Sitemaps, Storyboards, and Specifications: A Sketch of Website Design Practice. In *Proceedings of Designing Interactive Systems* (DIS 2000), 263-274.
6. Reitman, W.R. *Cognition and Thought*. John Wiley & Sons, Inc., 1965.
7. Sobek II, D.K., Ward, A.C., Liker, J.K. Toyota's principles of set-based concurrent engineering. *Sloan Management Review*, Winter 1999, 67-83.
8. Terry, M. and Mynatt, E.D. Recognizing creative needs in user interface design. In *Proceedings of the Fourth Conference on Creativity & Cognition*, 2002, 38-44.
9. Terry, M. and Mynatt, E.D. Side Views: Persistent, on-demand previews for open-ended tasks. In *Proceedings of UIST 2002*, 71-80.
10. Terry, M., and Mynatt, E.D. Variation in Element and Action: Supporting Simultaneous Development of Alternative Solutions. In *Proceedings of CHI 2004*, 711-718.
11. Verlinden, J.C., Igarashi, T., and Vergeest, J.S.M. Snapshots and Bookmarks as a Graphical Design History. In *Proceedings of International Design Conference 2002*, Dubrovnik, Kroatia, 567-572.
12. Ward, A., Liker, J.K., Cristiano, J.J., Sobek II, D.K. The second Toyota paradox: How delaying decisions can make better cars faster. *Sloan Management Review*, Spring 1995, 43-61.