Available online at www.sciencedirect.com

SCIENCE DIRECT®

Knowledge-Based Systems 18 (2005) 415–425

Knowledge-Based SYSTEMS

www.elsevier.com/locate/knosys

Enhancing general-purpose tools with multi-state previewing capabilities

Michael Terry*, Elizabeth D. Mynatt

Everyday Computing Lab, GVU Center, College of Computing, Georgia Tech, Atlanta, GA 30332, USA

Received 20 May 2005; accepted 15 July 2005
Available online 26 August 2005

Abstract

General-purpose design tools can be applied to a wide variety of design problems, but the large number of unique states they are able to produce makes it difficult to find results most relevant to a *specific* design problem. Current interfaces exacerbate this problem by offering only a single preview of one potential future state. We introduce *multi-state previewing tools* to facilitate the process of generating, displaying, navigating, and evaluating multiple, potential future states simultaneously. Multi-state previewing tools specifically encode and automate higher-level design practices, such as exploring multiple alternatives, better aligning computer-based tools with design. In this paper, we synthesize a framework for this class of tools by combining and generalizing existing instantiations, then show how this framework can be used to guide the design, implementation, and further research of these tools.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Multi-state previewing; Tool design; General-purpose tool

1. Introduction

Research in computer-based design tools has sought to assist the design process at various stages, and through varying levels of intervention. For example, capture and access services such as those provided by the Designer's Outpost [9] or Chameleon [24] offer peripheral support by capturing designers' interactions with natural media in the physical world. The design history that accumulates serves to document the process as it unfolds, but is also manipulable, enabling designers to modify past decisions and actions as needed. Other applications take a more central role and constitute the primary tool for solving portions of the design problem. For example, Denim [10] is an application that can be used as the main tool for the early phases of website design: using a pen-based interface, users can create and manipulate rough sketches of a website in Denim, then instruct the application to automatically translate these sketches into actual web pages.

This paper examines the process of using general-purpose computer-based tools to directly transform *existing data* within the design process. As such, we do not consider tools that support activities like free-form sketching (where the designer essentially begins with a 'blank slate'), but rather tools that take existing data, such as a digital image or sound recording, and modify that data to another, more desired form.

The specific problem we investigate is that of manipulating a general-purpose tool so that the states it outputs are those values most relevant to a particular design problem. A typical computer-based tool can transform a given input into one of millions of other possible values, all in a single step. Though this flexibility grants the user considerable power, current interfaces make it difficult for users to effectively wield this power in ways commensurate with design practices. For example, in the course of developing a solution, a designer may wish to compare and contrast multiple, potential solutions to determine which best solves the problem. However, current interfaces do not explicitly support the process of generating multiple alternatives simultaneously: applications allow a document to exist in only one state at a time, and offer no more than a single preview. This creates a form of 'interface tunnel vision' that limits the user's ability to quickly attain an overview of

* Corresponding author.
E-mail addresses: mterry@cc.gatech.edu (M. Terry), mynatt@cc.gatech.edu (E.D. Mynatt).

0950-7051/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2005.07.004

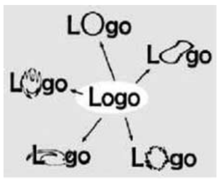

Fig. 1. User interface 'tunnel vision' limits the number of previews of potential future states. While a designer may wish to explore a handful of variations, she can preview only one at a time.

one's possibilities (Fig. 1). Consequently, users must develop their own workarounds to achieve desired practices.

To better align general-purpose design tools with common design practices, we offer *multi-state previewing tools*, a user interface mechanism that streamlines the process of generating, navigating, manipulating, and evaluating multiple, potential future states simultaneously. Multi-state previewing tools exploit modern-day computational power to generate previews of *sets* of possibilities, without requiring users to commit to any one value (Fig. 2). Conceptually, these tools help designers bridge two worlds within the design process: the set of future states acceptable to the designer and the set of states actually reachable with the computer-based tool (Fig. 3). Multi-state previewing tools help users create an intersection of these two spaces by initially displaying a broad sampling of possibilities, which can then be manipulated to show very selective portions (Fig. 4).

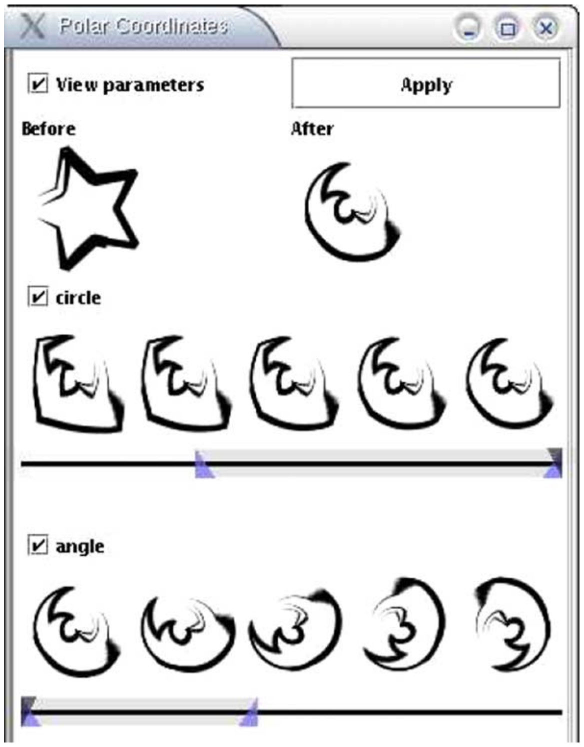

Fig. 2. Multi-state previewing tools extend the concept of a preview to show entire ranges of previews at one time. In this Side Views screenshot (a system described later), one can see ranges of previews for two of a command's parameters.

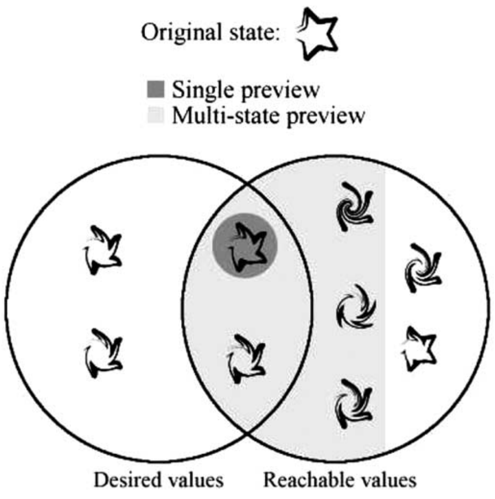

Fig. 3. Designers must intersect two worlds when using a general-purpose tool: the set of states applicable to a given problem (desirable states) and those reachable with the tool. Multi-state previews offer a more comprehensive view of the possibilities than single previews.

Multi-state previewing tools represent a theoretical concept derived from work we conducted to support expert users engaged in open-ended tasks [22,23]. We originally began this research by performing in situ observations of experts solving their everyday tasks. From these observations, we found that interfaces force users into linear modes of problem solving at odds with the non-linear, exploratory, and experimental practices

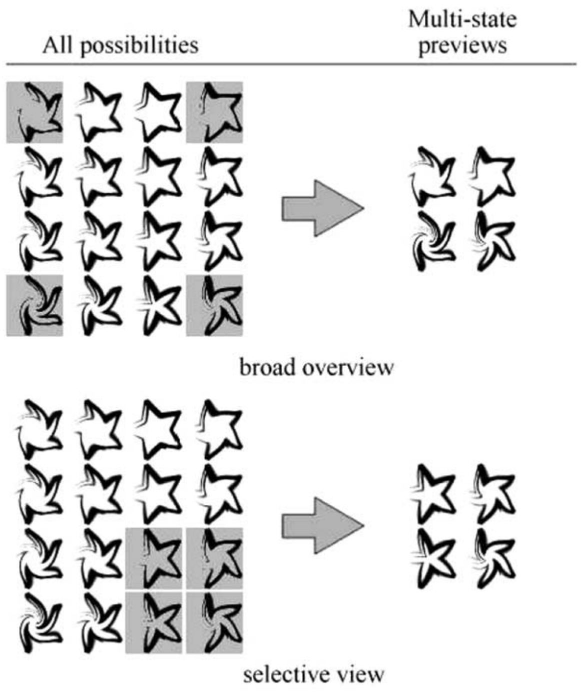

Fig. 4. A simplified depiction of multi-state previewing tools' capability to show broad or selective views of potential future states. The grids on the left represent all possible states from a given tool; shaded areas represent those states shown by the multi-state previewing tool on the right.

typical of design: designers would purposefully poke and prod the problem by testing out and experimenting with various commands, behaviors very similar to those described by Schön in his theory of reflection-in-action [19]. However, these activities were very rarely supported by specific user interface mechanisms, and thus cumbersome and time consuming for the user. As a result, we built Side Views [23], a system that enables users to quickly generate previews for multiple commands and parameters simultaneously.

Others have noticed the same deficiencies in user interfaces (e.g. Lunzer [11] and Marks et al. [13]), and have constructed tools to address these limitations. This paper represents the next step in this line work: it unites related efforts by creating a common theoretical framework for tools that offer users multiple previews, and offers a roadmap of what needs to be done next. The resultant framework suggests an alternative model of computing, from one requiring commitment to a single command for a task to progress, to one in which the user has the freedom to work with, contemplate, and embed multiple alternatives within their workspace. Multi-state previewing tools provide the first step towards this alternate vision of interaction; other work could advance this concept further by joining these types of tools with advanced history tools that more fully track the states a user has visited (e.g. as in [5,9]).

The framework also serves as a springboard for future efforts by identifying areas in need of further research. For example, to date there has been virtually no user testing of these tools; thus, while this class of tools can be easily motivated by comparing desired design practices with the practices afforded by current computer-based tools, there is still the unanswered question of how well these concepts play out in tools used for day-to-day design work.

The rest of this paper is structured as follows: first, we summarize characteristics of design problems and design practice, focusing on the large space of possibilities designers must consider when solving a design problem. We argue that the fluid, ever-changing nature of design problems implies a need for general-purpose tools, but observe that the large space of possibilities afforded by these tools adds to the burden of developing a solution. We then turn to a study we conducted that investigated the practices of expert users of an image manipulation application. The results from this study indicate that current interfaces offer considerable domain-specific functionality, but lack support for higher-level design practices. We then introduce multi-state previewing tools as a mechanism to help bridge the gap between design practices and computer-based tools. A set of existing multi-state previewing tools is reviewed, and a general framework for designing and implementing these tools is synthesized. We conclude by reflecting on opportunities for future work.

2. Design practice

Design problems have been variously described as ill-defined [17], ill-structured [6,20], and wicked problems [18]. While each of these terms describes a larger, more general class of problems, each is applicable to design, and each lends its own particular set of nuances and shades of meaning to the description of this activity.

Reitman was among the first to draw a distinction between well-defined and ill-defined problems, characterizing ill-defined problems as those lacking fixed goals and operators [17]. According to Reitman, ill-defined problems present a far greater challenge to solve than well-defined problems because the problem-solver must not only construct a solution, but must also define the problem and its boundaries. Simon [20] later questioned some of Reitman's assertions, arguing that problems that are supposedly well-defined by Reitman's definition exhibit characteristics of 'ill-definedness' when the search space exceeds the problem solver's capacities (such as available memory store). To illustrate his point, he gave the example of chess, a problem that is well-defined by Reitman's criteria, and claimed that choosing a move in a chess game is more like an ill-defined problem since, the search space is too large to create a perfect evaluation function for each and every move.

Unsatisfied with the broadening of the problem space implied by Simon's argument, Goel and Pirolli [6] drew a sharper distinction between design and non-design tasks by enumerating a set of characteristics more likely to be found in design task environments than in non-design task environments. In their definition of design problems, they reintroduce the notion of ill-defined goals, states, and operators as 'prototypical' features of design problems, and make the further requirement that something is actually *designed*, rather than simply solved.

While debate continues about what precisely defines and distinguishes design, there appears to be some consensus on particular characteristics of design problems and the practice of design. Goel and Pirolli [6] have identified a number of these common characteristics and lend support to their selections through a study of designers in three different disciplines (architecture, mechanical engineering, and instructional design). While the authors specifically do not claim that their criteria are complete and comprehensive, the properties identified do echo a number of other related research efforts (e.g. [17–20]), and thus serve as a useful starting point for discussing design. Rather than completely replicate their full set of claims, we summarize their findings with an eye towards those items most relevant to the theme of this paper (i.e. using computer-based tools to manipulate data within the design process).

In describing design *problems*, Goel and Pirolli assert that design problems:

- are large and complex with a number of interconnected parts that affect one another;

- have underdefined start and goal states, with equally underdefined transformation functions; and
- lead to solutions that can only be described as 'better or worse', as opposed to 'right or wrong'.

With regards to design *practice*, the authors found:

- distinct problem-solving phases (preliminary design, refinement, and detail design);
- problem structuring and decomposition;
- incremental development of the solution;
- the practice of limiting commitment to precise solutions; and
- personalized stopping rules and evaluation functions.

Again, many items in these lists summarize and draw upon previous work. For example, Rittel [18] and Reitman [17] have both observed that design problems do not have 'correct' solutions, just better or worse ones. However, what we would like to focus on is the notion that design problems create an enormous space of possibilities a designer can consider: not only can the designer develop multiple, equally viable solutions for one particular problem formulation, but she can also *renegotiate* and *redefine* the problem, thereby opening up an entirely new set of possibilities. Thus, one can view the designer's task as not only transforming the current state of affairs into a more desirable one [21], but also *pruning* the search space to consider only those possibilities most likely to yield favorable results. Many of the design practices suggest this type of activity: problem structuring defines the problem and sets its boundaries, preliminary design scopes out a rough solution, and so on. As we will see, this process of scoping down the problem space and finding solutions of interest can be either aided or hindered by general-purpose computer-based design tools, which themselves generate large numbers of possibilities to consider.

3. Computer-based tools for design

In this section we consider how features of design problems influence tool design, and argue that general-purpose tools will always be needed in the design process. We then review results from a study we conducted to understand how well some design practices are supported by current interfaces.

3.1. Implications for computer-based design tools

Just as solutions for design problems are never perfect, the features of design problems imply that there does not exist a 'perfect' design tool: at any moment an entirely new tool or method could arise for solving the problem in a better way. Similarly, the problem itself could be redefined in a way that favors one tool over another. As a real-world example, the comic industry is undergoing significant changes in the tools, techniques, and storytelling conventions it uses as comics move from print to the web [14]. With the move to the web, previous limitations imposed by printing technology and the cost of paper have been lifted. As a result, writers and illustrators are now expanding their repertoire of methods and tools for solving storytelling problems as the overall goal shifts from one of telling stories through the medium of paper, to telling stories through the medium of the web.

Given the malleability of design problems, it seems clear that general-purpose tools will always have a place in solving design problems, because they can, by definition, be applied to a large number of problems. This is a seemingly trivial point, but it has some important corollaries. First, what it does *not* imply is that automated computer-based tools—for example, tools that guide the designer through a series of steps (i.e. 'wizards') or tools that solve portions of the problem using some form of intelligence—have no place in the design process. Problem-specific tools can greatly reduce the time and effort needed to develop a solution for particular problem types. However, what it *does* suggest is that there is good reason to research how to augment general-purpose tools to make them better partners in the design process, since they will always play a pivotal role. That is, there are opportunities to improve the design process by enhancing general-purpose tools by means other than transforming them into special-purpose tools (Fig. 5). One such opportunity is facilitating the designer's process of manually fitting a tool to a particular problem.

Of the many states a general-purpose tool can generate, only a portion are applicable to a given problem. In ideal cases, the designer knows both the states desired and the method to manipulate the tool to generate those values. In these circumstances, the designer *searches* and *navigates* the space of possibilities to arrive at the desired values. When either the desired states or the method to arrive at

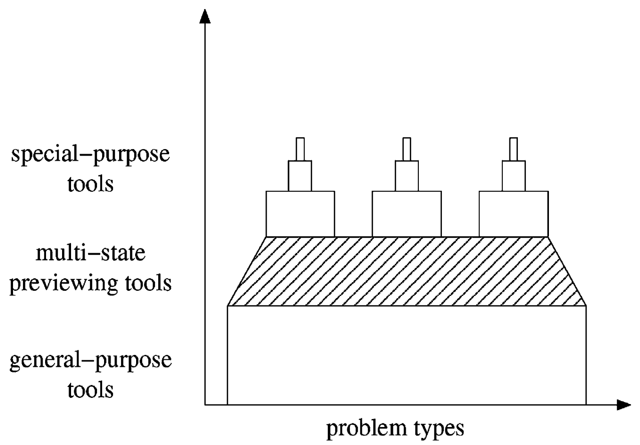

Fig. 5. The design process can be enhanced by creating specialized tools that are customized to a small number of problem types (the tall spires), or by augmenting general-purpose tools so that designers can more quickly fit them to particular problems (the shaded area).

those states is unknown, the task also becomes one of *experimenting* with the tool to develop an acceptable solution. In all cases, there is the possibility that more than one viable solution exists. Thus, the designer must continually *evaluate* the current problem and the potential solutions against the overall goals. Collectively, these activities—navigation, experimentation, and exploration—help the designer fit a tool to a particular problem instance. Throughout the rest of this paper, we will continue to frame our discussion of computer-based tools in terms of these themes of matching and constraining the set of states from a tool to those most relevant to the design problem.

3.2. Considering design practices using computer-based tools

To understand the problem solving process using computer-based tools, we interviewed and observed three expert users of an image manipulation application, an artist, a designer, and an image specialist for a major newspaper [22]. The interviews were structured to learn about the types of problems the subjects solved and the general problem-solving strategies they employed. Following the interview, subjects worked through a typical task of their choosing, such as designing an interface or toning (color-correcting) an image. While not all tasks were what one might consider 'design' tasks, they were all sufficiently open-ended and possessed most of the characteristics of design problems (e.g. they lacked well-defined evaluation functions, stopping rules, etc.).

Across the range of tasks we observed, the users engaged in many of the prototypical design practices noted above. However, very rarely were these practices explicitly supported by the application itself. Instead, users had to develop their own ad-hoc methods to achieve the desired functionality. In our study, we found users:

- employ the undo mechanism to probe the application's capabilities, and to experiment with the possibilities,
- 'scrub' sliders in dialog boxes to find, verify, and experiment with argument values,
- use undo and redo to evaluate a command after it had been applied, and
- make multiple copies of their data to support branching and limited commitment to particular solutions.

In some instances, the user would not know which commands or values would work best for a particular problem, and thus would repeatedly try and undo different commands to observe their effects. In other cases, the user was fairly confident about which command should be invoked, but less confident about which settings to apply. This uncertainty led to users 'scrubbing' sliders in modal dialog boxes, that is, making broad sweeps with slider controls that gradually narrowed around areas of interest.

Once a command had been applied, the designer would sometimes evaluate its effect by invoking undo and redo repeatedly, which had the effect of 'flashing' the two states—the current and most recent—on the screen. This practice allowed designers to compare two states in *time*, rather than side-by-side in *space*.

Finally, to limit commitment to particular choices, or to branch to explore alternatives, users would make copies of their data so they could more freely experiment with possibilities.

While these techniques helped the users cope with the complexities of solving their problems, at times the effort required was deemed too much, even though they may have improved the overall result. For example, in one instance we observed an artist coloring the clothes of a character in his painting, a task which involved manually applying paint strokes to the image. After painting the clothes using one shade of blue, he found the results less than satisfactory. However, he did not undo his work and try another shade because he did not feel it worth the effort to redo his paint strokes with another color. In this instance, the artist may have benefitted from tools that offered richer previews of the possibilities *before* he had to commit to actually painting the image.

Overall, we found that the application offers very sophisticated, domain-specific functionality, but generally lacks mechanisms to support higher-level problem solving practices. Within the scope of this paper, we found that users could not easily generate, find, experiment with, and evaluate multiple potential states: modal dialog boxes prevent previews of more than one command, and each command displays one preview at a time. Thus, side-by-side comparisons of commands are not possible when multiple, viable alternatives exist, and the single preview provides an impoverished view of the possibilities.

The reliance on modal dialog boxes with a single preview is one obvious deficiency in the interface's design. However, we believe there is a more fundamental problem with the way current interfaces are constructed: current interfaces do not cleanly support the non-linear, exploratory processes of design. While users may wish to create and contemplate multiple, potential solutions, applications honor only one state at a time and lock users into a linear model of interaction we term the *Single State Document Model*.

3.3. The Single Sate Document Model

The Single State Document Model (SSDM) is a model of interaction that requires a document to be in only one state at any point in time. Users progress through tasks by applying an operation, then working on the new state that results. While some past states are accessible via mechanisms like 'undo', this model allows only one to be active at a time.

Though conceptually simple for both user and implementor, this interaction model imposes a serial, linear progression through a task that is at odds with the design process. Thus, while a user may need to simultaneously consider multiple alternatives (for example, when she is uncertain which set of parameter values yield the 'best' result for a given command), the Single State Document Model creates a form of 'interface tunnel vision' that limits the number and types of views available at any one moment (Fig. 1). In essence, the SSDM favors a single, high fidelity WYSIWYG (What You See Is What You Get) view of the current document, when it is often just as useful to be able to view and compare a range of alternatives.

Multi-state previewing tools provide a conceptual shift from the SSDM to a model of interaction that recognizes a host of alternatives. We turn now to a description of this class of tools.

4. Multi-state previewing tools

Multi-state previewing tools are a class of user interface mechanism that enables a user to generate, navigate, manipulate, and evaluate multiple potential future states simultaneously, without needing to commit to any one particular value. Fig. 6 provides a comparison of the differences between normal previewing tools and multi-state previewing tools.

The capabilities of a multi-state previewing tool streamline many of the activities designers manually engage in with existing interfaces. First and foremost, the tools automatically generate and display a series of alternatives, a task that otherwise requires significant overhead and time on the user's part. Users can take advantage of these automated capabilities to help locate desired values, experiment without modifying the original data (making them perfect 'what-if' tools), and to perform side-by-side comparisons of several solutions. Use of these tools may also lead to the serendipitous discovery of viable alternatives: while searching for specific values of interest, the user may find a superior alternative she had not previously considered.

4.1. Specific implementations

A number of commercial and research systems have hinted at, or fully implemented, interfaces that offer users multiple previews. In this section, we first review systems that offer minimal support for this notion, then describe five research systems that are highly representative of this type of tool.

4.1.1. Systems offering basic support for multiple previews

See-Through Tools [3,4] (i.e. Toolglasses and Magic Lenses) are interface mechanisms that use the metaphor of a physical lens to represent functions within the user interface. Placing a lens over a document invokes that function on the data underneath and shows the results within the lens's frame. Multiple lens can be arranged side-by-side on the document at the same time, offering multiple previews. Additionally, lenses can be stacked on one another to combine the commands' effects. See-Through Tools provide basic support for multiple previews, but are somewhat limited by physical space (only so many lenses can be placed on a document at the same) and by the need to manually place the lenses on and off the document.

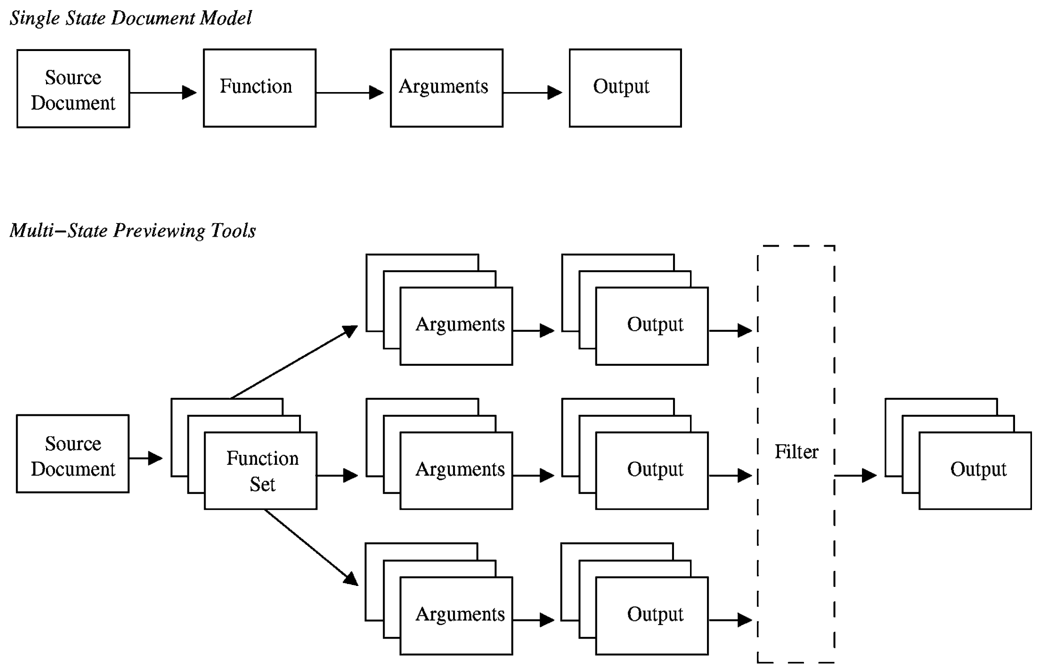

Fig. 6. Comparison of two previewing systems, the current convention of utilizing a single preview, and the proposed method of multiple previews.

Adobe Photoshop [1] includes a 'Variations' command that generates a ring of previews representing different color balance operations. Clicking on a preview has the effect of shifting the overall color balance by that amount, and updates other previews with the new value. This functionality offers a glimpse of the power of multiple previews, but the method of navigating possibilities—clicking on a preview—limits the ease with which users can quickly traverse the possibilities. Furthermore, this mechanism is available for only one command in the interface.

A common convention in interfaces for font selection is to provide a drop-down menu of font names, each rendered using its own font. Applications made by Corel [2], such as WordPerfect, employ this technique, and also preview the font on the current selection in the document. While this technique has obvious advantages over single previews, the concept has not been generally applied to other parts of the user interface.

From these basic systems we now turn to full-fledged multi-state previewing tools, beginning with the system we built, Side Views.

4.1.2. Side Views

Side Views [23] is a user interface mechanism that offers on-demand, persistent previews of one or more commands and their parameters. Side Views use the familiar tool-tip metaphor to provide a preview of a command within a pop-up window: when the user hovers the cursor over an object in the interface for a short period of time, a Side View appears with a preview of that command (Fig. 7). However, unlike a normal tool-tip, users can interact with the Side View and click on it to make it persist. Using this capability, multiple Side Views can be instantiated, allowing a user to directly compare and contrast multiple, potential commands at once (Fig. 8).

Side Views enable users to view and interact with a command's parameters through *parameter spectrums*, a series of previews for each parameter (Fig. 2). Initially, each parameter spectrum shows an evenly distributed sampling of values across the range of possibilities. However, users can modify the lower and upper bounds to make the range previewed as broad or narrow as desired. Collectively, the ability to instantiate previews for multiple commands and their parameters grants users a *breadth-first* view of the possibilities from their current state.

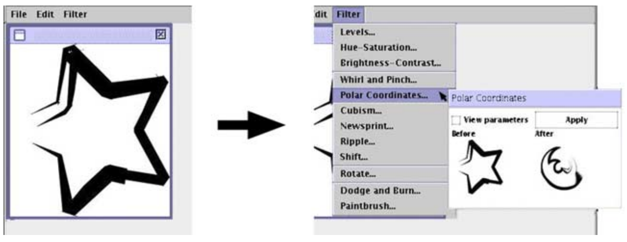

Fig. 7. Side Views use the familiar tool-tip metaphor to provide on-demand previews.

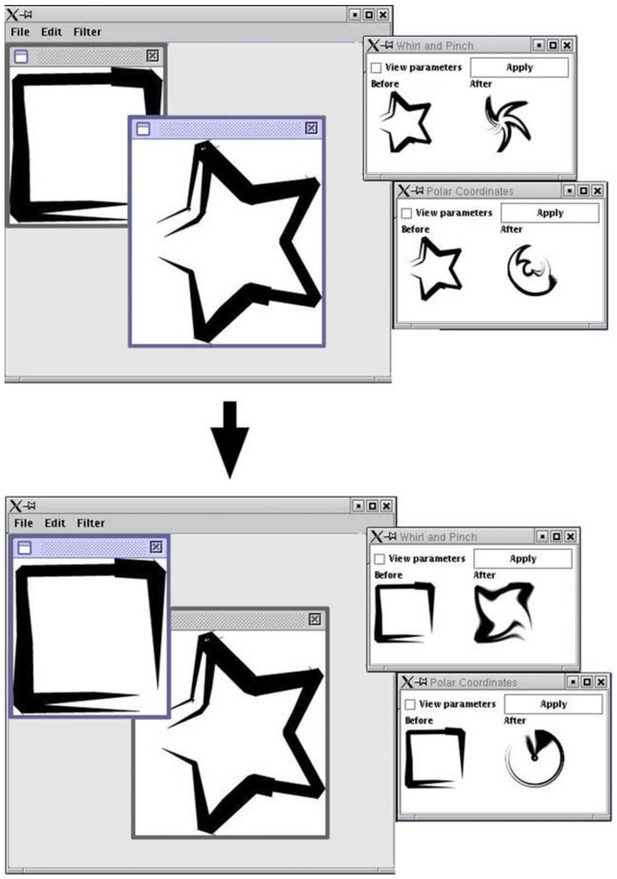

Fig. 8. Multiple Side Views can be instantiated at once, allowing comparisons *between* commands. These previews dynamically update as the active data changes, as in this example where the user switches the current document.

To gain a *depth-first* view of the possibilities, multiple Side Views can be combined by dragging and dropping one command's preview on another to create a chain of commands. Like a single Side View, combined Side Views show previews for each command and its parameters, with the left-to-right ordering of the commands indicating the order of operations (Fig. 9). Changes made to left-most commands ripple down the chain, allowing users to view how settings in earlier commands affect later commands.

Side Views also offer previews of direct user input, such as mouse input. For example, a user can instantiate a Side View for the paintbrush, then mouse over the active document. As the user moves the cursor over the document, the Side View updates its preview to include a paint stroke corresponding to the mouse movement. Users can then interactively vary the parameters for the paint stroke within the Side View, tweaking the characteristics of the tool until it meets the users' needs, all without ever modifying the current document.

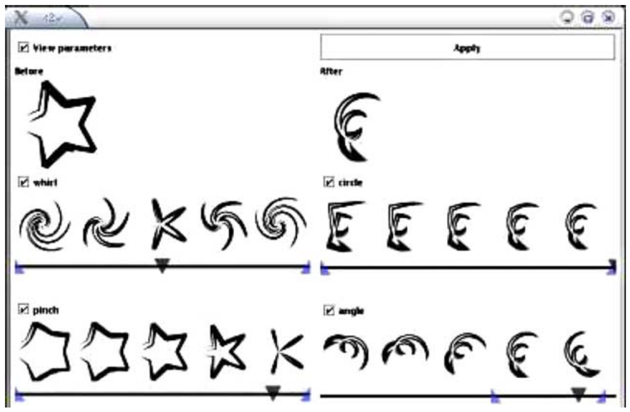

Fig. 9. Multiple commands can be combined in Side Views to form a chain of commands. Here, we see two commands and their parameters that are sequentially applied to the current document.

4.1.3. Design Galleries

Design Galleries [13] is a multi-state previewing system that varies a set of commands and parameters to produce a set of semantically distinct output. While most other instances of multi-state previewing tools present relatively raw, unfiltered previews of commands, Design Galleries explicitly attempts to facilitate the process of finding values of interest by filtering its output to display only those states that differ most from one another.

Originally, Design Galleries was applied to graphics applications, such as 3D modeling programs. In these applications, the system varies parameters, such as the type of light used to illuminate a 3D scene, then culls the output to display only those scenes that are the most perceptually different from one another. A later application investigated the use of this same technique in the design of radio antennae [16].

One of the noteworthy aspects of Design Galleries is that it re-conceptualizes the process of operating on data from one that is very computer-centric (i.e. choose a function, choose values for the function’s parameters) to one that is more closely aligned with how designers work through design problems (i.e. generate multiple alternatives, and compare and contrast them). One can imagine applying this concept to a full-fledged application, so that the user does not seek out commands to invoke, but instead views the possibilities offered by the tool, chooses one or more promising states, refines the selection, and repeats the process.

In their description of the Design Galleries, Mark et al. describe five core features of the system: an input vector (essentially, the parameters to vary); a ‘dispersion method’ that generates a set of input vectors which produce widely varying output; a function that applies the input vector to the current state to produce a set of results (what they call a mapping process); a ‘distance metric’ that evaluates the similarity between the various output; and a display that presents the output to the user. We will build on many of these same concepts when we construct a general framework for multi-state previewing tools.

4.1.4. Spreadsheet-like interfaces

In the spreadsheet-like information visualization system built by Jankun-Kelly and Ma [8], users can view areas of a data set by choosing parameters of interest and placing them in the left-most column and top-most row of a tabular interface. Users can also enter scripts in cells to programmatically control the parameter values displayed. With the desired parameters in place, every other cell in the table renders a slice of the data set using the parameters chosen for its particular row and column. To enhance this visualization, results can also be animated.

While this system has been applied to information visualization problems (i.e. large data sets of multiple dimensions), the same concepts could be applied to solving design problems. In particular, the notion of being able to enter scripts to control the output and the ability to animate the results could both have practical application in other multi-state previewing tools.

4.1.5. Subjunctive interfaces

The subjunctive interface is a concept in which users select multiple, tentative settings for commands and controls, then view the outcomes of those settings overlaid on one another in the same space [11,12]. For example, in a cannon simulator users can select multiple parameter values for objects within the simulation (e.g. different cannon angles), then run the simulation to see all possibilities execute simultaneously [11]. A more sophisticated example of this concept echoes user input, such as pointer input, across multiple copies of the same data, in real-time [12]. Each copy interprets the input differently, enabling the user to perform input once, then choose the most appropriate interpretation of that input. For example, in one application the user has the task of selecting an object in a medical image. As the user makes the selection, one document copy shows the ‘raw,’ unaltered selection, while another uses an intelligent system to guess the intended selection.

4.1.6. Suggestive interfaces

Suggestive interfaces [7] are systems that suggest potential future states by inferring intentions from interactions within a direct manipulation interface. Chateau, the 3D modeling application illustrating this concept, allows users to construct 3D models using line segments drawn in a 3D space. As the user creates or selects segments, the system analyzes the current context then generates a set of potential future states logically incorporating those elements. If one of the future states matches the user’s intentions, she can click on it to commit to that state.

4.2. A framework for multi-state previewing tools

From the range of tools presented, we are now in a position to construct a general framework for this class of tools. Building from the architecture of Design Galleries, we can generalize to the following set of components for a multi-state previewing tool:

1. a method that (manually or automatically) chooses the command or set of commands to be explored,
2. a method that (manually or automatically) produces a set of arguments for the chosen commands' parameters,
3. a mechanism that automatically iterates over the set of commands and their respective arguments, applying each command to its respective argument set,
4. an optional filter to constrain the output that results,
5. a method to present the results to the user,
6. a mechanism to refine the command and argument values chosen earlier, and
7. a mechanism that enables the user to choose a given output.

4.2.1. Command selection

Command selection represents the process of determining which commands are most relevant to the current problem state. This process can be a manual, user-driven process (i.e. the user chooses the commands of interest), or it can be automated (the application uses the given context to suggest one or more commands of potential interest to the user). Side Views, the spreadsheet-interface, and the Subjunctive Interface use a manually-driven command selection process, while Design Galleries and Chateau's command selection processes are automated.

When considering commands that may be previewed using multi-state previewing tools, it is important to keep in mind that commands may also be invoked as a result of directly manipulating the interface, as opposed to selecting a menu item or pushing a button. While a tool like Side Views clearly caters to commands invoked through menus and toolbars, it is just as conceivable to produce multiple previews when interacting through direct manipulation, as evidenced in Chateau.

4.2.2. Argument selection

Argument selection is the process of choosing sets of argument values for the commands selected in the first step. As with command selection, this process may occur manually, or automatically. For example, when using Side Views, users manually designate valid argument sets by varying the lower and upper bounds of the values previewed by parameter spectrums; Design Galleries employ heuristics to determine these values.

4.2.3. Iterative function application

Once commands and their argument sets have been selected, a multi-state previewing tool systematically invokes each command/argument pair on a copy of the active data to generate a set of output. This output is then passed on to an optional filter.

4.2.4. Output filter

An output filter constrains the set of results generated to display only a subset of these values. For example, Design Galleries includes a fairly sophisticated filter that limits the set of results to those that differ most from one another.

4.2.5. Presentation mechanism

The results of generating and filtering previews are made available to the user through a presentation mechanism. Though the output may have been filtered somewhat in the previous step, there are still opportunities to filter or highlight the results using information visualization techniques. For example, the tool may opt to show as many different results as possible, or attempt to draw attention to those it thinks are most applicable to the current context.

4.2.6. Refinement

When the process of generating and displaying multiple previews has finished, the user then has the opportunity to refine the set of values previewed, essentially repeating any of the prior steps. This is the moment at which the user takes advantage of the multiple previews to compare and contrast alternatives, explore, and experiment with the possibilities.

4.2.7. Selection

After searching the space of possibilities, the user can then choose one of the output to replace the current state (the equivalent of actually applying a command and its arguments).

4.3. Reflecting on the framework

The components comprising a multi-state previewing tool provide an idealized view of these tools: while every instantiation can be said to implement each component in some form, they may not be cleanly separated nor configurable at runtime. For example, the process of choosing commands and arguments is merged into one automated process in Chateau. However, the framework illustrates areas that can be customized by either the tool designer or the actual end-user. For example, it is conceivable that the tool provides the capability for the end-user to write her own output filter so that it constrains its results to a particular range of values.

5. Summary and opportunities for future work

In this paper, we have argued for a new class of user interface mechanism called multi-state previewing tools, and motivated their need by contrasting the practices employed by designers and those afforded by current interfaces. Multi-state previewing tools are intended to support and streamline the experimental, exploratory, evaluative, and iterative practices of design by explicitly encoding parts of these processes in the user interface. The framework offered provides a guide to building these tools, divides the tools into logical components that can be individually studied and enhanced, and offers a vocabulary and structure to facilitate evaluations and comparisons.

The framework also suggests areas in need of future research. While the basic concepts have been demonstrated by a number of tools, there are several areas that are particularly under-researched at this moment.

The multi-state previewing tools built thus far have dealt primarily with data that are static and visual in nature (the Design Gallery's antenna design application is an exception). Opportunities remain to apply these concepts to non-visual domains, such as sound manipulation, and to time-based data, such as video.

Another area that has not been thoroughly investigated is the tight integration of multi-state previewing tools with direct manipulation interfaces. The Chateau application and Subjunctive Interfaces come closest to providing previews within direct manipulation interfaces, but more opportunities exist to understand how to apply these concepts to other operations, such as drag-and-drop. For example, when users drag and drop a file in a file explorer, holding down a modifier key can copy the file, rather than move it. However, this option is essentially hidden to the user, and must be known *a priori*. Multi-state previewing tools could be applied in this situation to offer users previews of both possibilities, so that they could manually select one or the other if they did not know which modifier key to press.

The design of multi-state previewing tools faces many of the same challenges as the design of information visualization tools: choosing multiple commands and sets of parameters creates a multi-dimensional space of results that cannot be displayed all at once. The challenges, then, are to display these results in ways most useful to users, while providing them with tools that let them fluidly navigate the space of possibilities. More so than with information visualization tools, these tools must integrate well with the main interface and the user's workflow, because these tools augment the user's primary tool and need to work in concert with it.

Another design challenge shared with information visualization tools is the timely delivery of the previews. Current implementations dynamically generate their results, and in some cases, this can literally take hours (as with Design Galleries). Therefore, there is an important question regarding the relationship between the number of previews, the time required to generate them, and the fidelity required for the previewing system to be maximally beneficial. For example, it may be the case that users can achieve the best results when a few rough previews are generated quickly as opposed to the system creating more, higher-fidelity sets of previews. To date, none these issues have been explored nor evaluated.

Of the systems reviewed, only the spreadsheet-like interface offers the option for end-user programming. As has been argued elsewhere [15], end-user programming facilities enable users to customize a general-purpose tool so it better matches a user's particular problem domain; offering this capability in multi-state previewing tools could provide similar benefits.

Finally, the area most in need of research is the evaluation of these tools: no long-term testing has been performed with any existing system, nor have there been any sizable user studies. Multi-state previewing tools represent a conceptual shift in how the interface should assist users in solving a task: current interfaces assume the user only needs to develop a single solution by sequentially applying commands, while multi-state previewing tools do not make this assumption. Instead, multi-state previewing tools begin with the premise that regardless of whether the user knows what she wants or not, she will most probably need to contemplate multiple alternatives before committing to any one in particular. The instantiation of this concept in user interface mechanisms is largely untested, and should be the next step in understanding the suitability of these types of tools in the design process.

References

[1] Adobe Photoshop. http://www.adobe.com.
[2] Corel Corporation. http://www.corel.com.
[3] E.A. Bier, M.C. Stone, K. Fishkin, W. Buxton, T. Baudel, A taxonomy of see-through tools Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, 1994. pp. 358–364.
[4] E.A. Bier, M.C. Stone, K. Pier, W. Buxton, T.D. DeRose, Toolglass and magic lenses: the see-through interface, Computer Graphics 27 (1993) 73–80.
[5] W.K. Edwards, E.D. Mynatt, Timewarp: techniques for autonomous collaboration Conference Proceedings on Human Factors in Computing Systems, ACM Press, New York, 1997. pp. 218–225.
[6] V. Goel, P. Pirolli, The structure of design problem spaces, Cognitive Science 16 (3) (1992) 395–429.
[7] T. Igarashi, J.F. Hughes, A suggestive interface for 3D drawing Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology, ACM Press, New York, 2001. pp. 173–181.
[8] T.J. Jankun-Kelly, K.-L. Ma, Visualization exploration and encapsulation via a spreadsheet-like interface, IEEE Transactions on Visualization and Computer Graphics 7 (3) (2001) 275–287.
[9] S.R. Klemmer, M. Thomsen, E.P. Goodman, R. Lee, J.A. Landay, Where do web sites come from? Capturing and interacting with design history Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, 2002. pp. 1–8.
[10] J. Lin, M.W. Newman, J.I. Hong, J.A. Landay, Denim: Finding a tighter fit between tools and practice for web site design Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, 2000. pp. 510–517.
[11] A. Lunzer, Towards the subjunctive interface: general support for parameter exploration by overlaying alternative application states Late Breaking Hot Topics, IEEE Visualization '98, 1998 pp. 45–48.
[12] A. Lunzer. Choice and comparison where the user wants them: subjunctive interfaces for computer-supported exploration. In Proceedings of IFIP TC, 13 International Conference on Human-Computer Interaction (INTERACT '99), pp. 474–472, 1999.
[13] J. Marks, B. Andalman, P.A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall,

J. Seims, S. Shieber, Design galleries: a general approach to setting parameters for computer graphics and animation Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press/Addison-Wesley Publishing Co., New York, 1997. pp. 389–400.

[14] S. McCloud, McCloud, Reinventing Comics: How Imagination and Technology are Revolutionizing an Art Form, Perennial, UK, 2000.

[15] B.A. Nardi, A Small Matter of Programming: Perspectives on End User Computing, MIT Press, Cambridge, MA, 1993.

[16] A. Quigley, D.L. Leigh, N.B. Lesh, J.W. Marks, K. Ryall, K. Wittenburg. Semi-automatic antenna design via sampling and visualization. In IEEE AP-S International Symposium and USN-C/URSI National Radio Science Meeting (APS/URSI), 2002.

[17] W.R. Reitman, Reitman, Cognition and Thought, Wiley, NY, USA, 1965.

[18] H.W.J. Rittel, M.M. Webber, Developments in design methodology, Planning Problems are Wicked Problems (1984) 135–144.

[19] D. Schön, Schön., The Reflective Practioner: How Professionals Think in Action, Basic Books, New York, NY, 1983.

[20] H. Simon, Simon, The structure of ill-structured problems, Artifical Intelligence 4 (1973) 181–203.

[21] H.A. Simon, The Sciences of the Artificial, MIT Press, Cambridge, MA, 1969.

[22] M. Terry, E.D. Mynatt, Recognizing creative needs in user interface design Proceedings of the Fourth Conference on Creativity and Cognition, ACM Press, New York, 2002. pp. 38–44.

[23] M. Terry, E.D. Mynatt, Side views: persistent, on-demand previews for open-ended tasks Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology, ACM Press, New York, 2002. pp. 71–80.

[24] M. Tsang, G.W. Fitzmaurice, G. Kurtenbach, A. Khan, B. Buxton, Boom chameleon: simultaneous capture of 3D viewpoint, voice and gesture annotations on a spatially-aware display Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology, ACM Press, New York, 2002. pp. 111–120.