

# Grizzly Bear: A Demonstrational Learning Tool for a User Interface Specification Language

Martin R. Frank

Graphics, Visualization & Usability Center  
Georgia Institute of Technology  
Atlanta, Georgia 30332-0280  
martin@cc.gatech.edu

## ABSTRACT

Grizzly Bear is a new demonstrational tool for specifying user interface behavior. It can handle multiple application windows, dynamic object instantiation and deletion, changes to any object attribute, and operations on sets of objects. It enables designers to experiment with rubber-banding, deletion by dragging to a trashcan and many other interactive techniques. To the author's best knowledge it is currently the most complete demonstrational user interface design tool that does not base its inferencing on rule-based guessing.

There are inherent limitations to the range of user interfaces that can ever be built by demonstration alone. Grizzly Bear is therefore designed to work hand-in-hand with a user interface specification language called the Elements, Events & Transitions model. As designers demonstrate behavior, they can watch Grizzly Bear incrementally build the corresponding textual specification, letting them learn the language on the fly. They can then apply their knowledge by modifying Grizzly Bear's textual inferences, which reduces the need for repetitive demonstrations and provides an escape mechanism for behavior that cannot be demonstrated.

**KEYWORDS:** Rapid prototyping, user interface specification languages, programming by demonstration.

## INTRODUCTION

Grizzly Bear is the bigger brother of a previous system called Inference Bear [1,2]. The behavior that can be demonstrated to Grizzly Bear is a clean superset over that of the earlier system. The enhancements include conditional behavior as well as behavior exhibited by sets of objects.

This higher expressive power is achieved by incorporating *positive* and *negative* examples in addition to the *before* and *after* examples already used by Inference Bear. Figure 1 shows Grizzly Bear's user interface.

Giving one example to Grizzly Bear consists of working through its iconic buttons from left to right. One demonstration consists of one or more such examples. After each example, the corresponding textual inference is displayed, and the behavior can be tested interactively by temporarily going into a test drive mode.

*Cite as:* Martin R. Frank. Grizzly Bear: A demonstrational learning tool for a user interface specification language. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 75-76, (Pittsburgh, Pennsylvania, November 15-17) 1995.

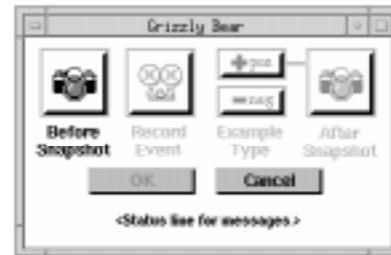


Figure 1. Grizzly Bear's Control Panel.

The use of Grizzly Bear is best explained by examples. The rest of this paper shows how to build a small "file manager" application which lets users create and delete documents and folders. Figure 2 shows its user interface, which is laid out using a conventional user interface builder [3].

The behavior to be demonstrated is that new folders and documents can be created on the main canvas by dragging from their prototypes. They can then be moved around, and they can be deleted by dragging them to the trashcan.

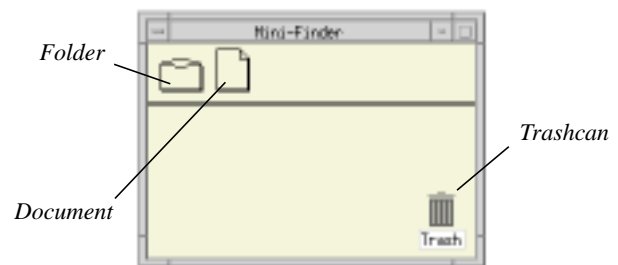


Figure 2. Layout of the Mini-Finder Application.

We will discuss creation by dragging in detail. Assume the designer decides that pressing the mouse down on the *Folder* icon makes a new red folder appear under the mouse, that the new folder then tracks the mouse, and that releasing the mouse button stops the tracking and makes the folder black.

Showing this functionality consists of one demonstration each for the *press*, *motion* and *release* events on the *Folder* icon. (All events between a *press* and a *release* event are received by the element which received the *press* event.)

The first demonstration shows Grizzly Bear that pressing down on the *Folder* icon makes a new red folder appear there (Figure 3). The *before* snapshot is on the left, with a feedback icon representing the triggering event superimposed. (The feedback icons are borrowed from Marquise [4].) The *after* snapshot is on the right. In this case, the *after* snapshot shows that a red copy of the prototype folder is created (the red color appears grayish in the screen shot).



Figure 3. Drag-and-drop: the *Press* Event.

If we were to test the behavior now, we could create a new folder. However, it would not follow the mouse but rather appear over the prototype folder, obscuring it.

So let us show that the newly created folder icon then follows the mouse. This can be done by demonstrating that red objects follow the mouse, as the red color is one way of telling the newly created folder apart from the others. We demonstrate this behavior by introducing two temporary folder icons in the first *before* snapshot, and by then demonstrating that the red one follows the mouse (Figure 4). This demonstration consists of two examples.



Figure 4. Drag-and-drop: the *Motion* Event.

If we test the current behavior now, we can create the first folder as desired, but it appears in red on the canvas. If we create further new folders, we will drag the previous folders as well as the new ones! This is the case because Grizzly has inferred that all red objects track the mouse, and because all folders remain red after their creation so far.

Let us thus complete the example by demonstrating that the color of newly created folders reverts to black upon the *release* event of the drag-and-drop interaction (Figure 5).



Figure 5. Drag-and-drop: the *Release* Event.

If we test the current behavior now we can indeed create folders by dragging from the palette as intended.

We can only outline how the remaining functionality is demonstrated due to space limitations. First, we want to be able to create documents in the same way as folders. We can achieve this by repeating the demonstrations above for documents, or we can make copies of the generated textual specifications for folders, and adapt them for documents. (For the sake of completeness, we could also generalize the synthesized specifications to work for both folders and documents, or we could have demonstrated the more general behavior in the first place. However, these solutions require more sophistication on the designer's part.) The remaining functionality concerns moving and deleting objects. These demonstrations make use of negative examples for the first time in this paper. We show that canvas objects follow the mouse (positive examples) while the prototype objects and the trashcan do not (negative examples). Similarly, in the final demonstration, releasing canvas objects over the trashcan deletes them (positive examples), but this is not true for the prototype objects (negative examples).

## CONCLUSION

Grizzly Bear contributes to the state of the art by expanding the range of user interfaces that can be demonstrated without resorting to rule-based guessing as an inferencing strategy. It differs from existing demonstrational systems in its role as primarily a learning tool for the underlying specification language - allowing designers to get started quickly without being later limited to behavior that can be demonstrated.

## ACKNOWLEDGEMENTS

Brad Myers contributed the feedback icons for triggering events. The camera and recorder icons were designed by Kevin Mullet while at Sun Microsystems.

## REFERENCES

- [1] M. Frank and J. Foley. A pure reasoning engine for programming by demonstration. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 95–101, (Marina del Rey, CA, Nov. 2-4) 1994.
- [2] M. Frank, P. Sukaviriya, and J. Foley. Inference Bear: Designing interactive interfaces through before and after snapshots. In *Proceedings of the ACM Symposium on Designing Interactive Systems*, pages 167–175, (Ann Arbor, MI, Aug. 23-25) 1995.
- [3] T. Kuehme and M. Schneider-Hufschmidt. SX/Tools - An open design environment for adaptable multimedia user interfaces. *Computer Graphics Forum*, 11(3):93–105, Sep. 1992.
- [4] B. Myers, R. McDaniel, and D. Kosbie. Marquise: Creating complete user interfaces by demonstration. In *Proceedings of INTERCHI, ACM Conference on Human Factors in Computing Systems*, pages 293–300, (Amsterdam, The Netherlands, Apr. 24-29) 1993.