INFERENCE BEAR: INFERRING BEHAVIOR FROM BEFORE AND AFTER SNAPSHOTS

Martin R. Frank James D. Foley
{martin,foley}@cc.gatech.edu

Graphics, Visualization & Usability Center
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

ABSTRACT

We present Inference Bear (Inference Based On Before And After Snapshots) which lets users build functional graphical user interfaces by demonstration. Inference Bear is the first Programming By Demonstration system based on the abstract inference engine described in [5].

Among other things, Inference Bear lets you align, center, move, resize, create and delete user interface elements by demonstration. Its most notable feature is that it does not use domain knowledge in its inferencing.

INTRODUCTION

We use a technique for demonstrating behavior which is inspired by the "before" and "after" pictures often found in advertisements. Behavior is demonstrated by supplying a Before picture, an After picture and an event that causes the transition from the Before to the After picture. A single example is sufficient for simple behavior, more examples are required to infer more complex behavior.

Snapshots have been used in earlier demonstrational systems, such as [7], which lets the user take snapshots of valid states. Before and After snapshots add sequentiality to the demonstration.

Inference Bear is build on top of two main components. The first component is an abstract demonstrational inference engine [5], the second one is an interactive user interface builder [6]. Consequently, Inference Bear only adds a modest amount of additional code which combines these.

PROGRAMMING BY DEMONSTRATION

We shortly review previous work. Peridot [9] supports designing scrollbars, buttons, choice boxes and similar objects by demonstration. Lapidary [10] focuses on creating application-specific objects. Metamouse [8] learns graphical procedures by example. Druid [13] lets users attach simple functionality such as enabling, disabling, hiding and showing to buttons. Eager [1] watches users perform operations and detects and automates repetition. DEMO [14,2] uses a stimulus-response paradigm for demonstrating the behavior of graphical objects. Chimera [7] infers constraints between graphical objects given multiple snapshots. Finally, Marquise [11] uses domain knowledge in order to support building graphical editors.

This is "Frank, M. and J. Foley, *Inferring Behavior From Before and After Snapshots*, Technical Report git-gvu-94-12, Georgia Institute of Technology, Graphics, Visualization & Usability Center, April 1994."

All of these systems use by-demonstration techniques but they are not easily compared because they have different goals and use different techniques. Nevertheless, we make an attempt to classify them in Table 1.

The first two columns describe user interface aspects. The first column shows if the system is constantly watching the user during normal operation or if it is explicitly invoked. The advantage of constant watching is that the users do not have to learn anything new to take advantage of the system. Some systems query the user when they make an inference (marked with "Query" in the table), others indicate their inferences by subtly displaying their prediction of what the user is doing next ("Prediction"). The second column describes if the system asks the user for confirmation and clarification after each inference. Any inferencing system will sometimes guess wrong - the clarification dialog gives the user an opportunity to correct and fine-tune inferred behavior. The disadvantage here is that going through this clarification process after every inference can be distracting.

The next two columns make an attempt to measure the capability of the systems. The third column indicates if the prototypes can handle the creation and deletion of graphical objects at run-time. The fourth column indicates to what degree the systems can infer geometric relationships between objects. This classification is inherently subjective because the design goals of these systems are different (for example, Eager does not attempt to infer geometric relationships) and because their capabilities are different (for example, Eager is "stronger" than Inference Bear in being able to deal with repetition, Inference Bear is "stronger" in inferring relationships). We labelled systems which can detect simple relationships such as centering and touching "Low", systems which can detect more general relationships "Medium" and the most sophisticated systems "High". This, again, is a crude and necessarily subjective classification.

The remaining columns are concerned with the implementation of the inferencing systems. The fifth column describes

	Interface Is Eager (Constantly Watches User)	*Interface* Uses A Clarification Dialog	*Capabilities* Dynamic Object Creation + Deletion	*Capabilities* (Subjective) Strength in Geometric Relations	*Internals* Search Space Reduction	*Internals* Is Rule-Based	*Internals* Temporary Behavior Storage	*Internals* Output
Peridot [9] 1987	Yes (Query)	Yes	No	Low	None	Yes	Snapshots	One-Way Constraints
Lapidary [10] 1989	No	Yes	No	Low	None	No	Snapshots	One-Way Constraints
Metamouse [8] 1989	Yes (Prediction)	No	No	Medium	Explicit[a] (Aux. Objs)	No	not applicable	Graphical Procedure
Druid[b] [13] 1990	No	Yes	No	None	None	No	Event Recording	Script
Eager [1] 1991	Yes (Prediction)	No	not applicable	not applicable	not applicable	No	Event Recording	Macro
DEMO [2,14] 1991/92	No	Yes	Yes	High	Explicit (Aux. Objs)	Yes (DEMO II)	Compressed Snapshots	Response Description
Chimera [7] 1991	No	No	No	High	Explicit[c] (Aux. Objs)	No	Snapshots	Two-Way Constraints
Marquise [11] 1993	No	Optional[d]	Yes	Low	None	Yes	Event Recording	LISP Code
Inference Bear 1994	No	No	Yes	Medium	Implicit	No	Ev. Rec. and Snapshots	Script

Table 1. Overview of Demonstrational Systems

a. Metamouse additionally reduces the search space by only considering touch relations.
b. Note that only Druid's demonstrational component is discussed here but not its rule-based design assistant.
c. Chimera also uses a variety of other techniques to reduce the solution cost of a demonstration.
d. A feedback window is displayed; the designer can change aspects of the behavior but does not have to.

if the system reduces the number of objects that it checks for relationships. Some systems use auxiliary objects such as guide wires to let the user specify the relevant objects and their relationship. Inference Bear limits the number of objects it considers implicitly but not heuristically [5]. The sixth column describes if the inferencing is based on rules or on an algorithm. The advantage of rule-based systems is that they can encode commonly used behavior. The disadvantage is that the rules can sometimes miss even simple relationships while algorithm-based systems can handle all relationships within a certain class. The seventh column describes how previous demonstrations are captured. There are two main approaches towards capturing demonstrations. Event-recording stores the events during a demonstration by the user. Snapshot-taking records a series of states. The last column describes the output of the inferencing process.

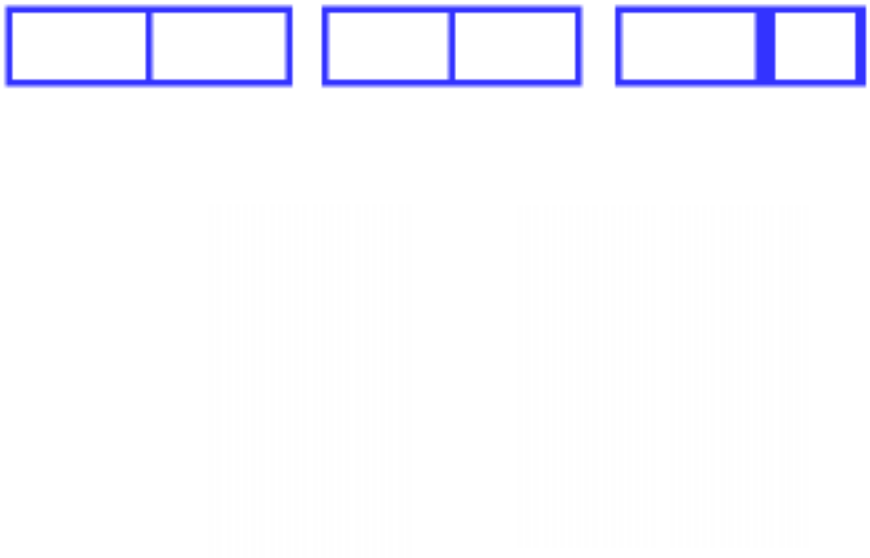


Figure 1. Overview of the Design Environment.

THE DESIGN ENVIRONMENT

Figure 1 shows an overview of our design environment. The upper three windows with the dark background are the control panels of its main components.

The left-hand control panel belongs to the *interface builder* we are using [6]. It allows to open toolboxes and user interface designs. Two such toolboxes are shown on the left (mislabelled "Adapter"). The current user interface design is shown in the middle of the figure (labelled "Design Mode"). In this example, the design consists of two buttons *a* and *b* which are connected by line *line*.

The middle control panel belongs to *UIDE*, the User Interface Design Environment [3]. In the context of this paper, it suffices to know that it lets you open the two text editors shown to the right and that it can switch the current design to run mode by reading and executing their specifications. We will ignore the "Application Model" editor in this paper. The "Interface Model" editor specifies the behavior of the interface design at run time. In Figure 1, it specifies how the line between *a* and *b* changes when *b* is moved.

Finally, the right-hand control panel provides the interface to advanced *tools* that the designer can use in this environment. The "Interview Tool" helps the designer fill in the textual specifications by asking questions based on the current user interface layout. It is described in [4] and will not be further discussed here. The "Inference Bear" is the tool we are concerned with - it lets designers describe the behavior of the user interface by demonstration. For example, it can infer the specification shown in the Interface Model editor of Figure 1.

INFERENCE BEAR

Figure 2 shows the control panel that appears when the user clicks on the "Inference Bear" button in the "Tools" window.

Figure 2. Inference Bear

Giving one example consists of working through the four iconic buttons from left to right. The designer first sets up for the "Before Snapshot" by editing the current design (using the interface builder). She then clicks the "Before snapshot" button to tell the system that she is done.

The next step is to tell the system to what kind of event the behavior should be attached to. This is done by first clicking the "Start Recording" button which switches the interface builder to event recording mode. The designer then causes an event in the current interface design, such as a "move", "click" or "resize" event. She then clicks the "Stop Recording" button which switches the interface builder back into the standard editing mode.

The final step is to tell the system what should happen if the interface is in the state given by the Before snapshot and this event occurs. This is done by bringing the user interface de-

sign to the state it should go to, and by pressing the “After Snapshot” button to let the system know that one is done.

The system now responds by running the inference engine [5] on this example. The designer than tests if the inferred behavior is the one she had in mind by either reading the inferred specification or by going to Run mode and testing it interactively. If it is acceptable, she clicks “OK” in Inference Bear’s control panel which causes the inferred behavior to become part of the overall Interface Model.

If it is not what she had in mind, she proceeds by giving another example in the manner described above. The inference engine is then called using the old examples plus the new one. The designer now again chooses if she is satisfied with the new behavior or wants to give more examples.

EXAMPLE 1: A COMPLETE INTRODUCTORY EXAMPLE

Let us explain this dialog between the designer and Inference Bear with a simple example. Assume the designer has created a user interface layout consisting of a window containing a single button. The behavior to be inferred is that the button moves one button length to the right every time it is clicked on.

She then calls the Inference Bear and provides a first example which consists of the Before and After snapshot shown in Figure 3. The recorded event is a click on the button.

Figure 3. The first example. The Before snapshot is shown above, the After snapshot below.

The system responds by showing the script in Figure 4.[1].

Inference Bear’s initial conjecture here is that the button moves to the absolute position shown in the After snapshot. The designer thus has to give another example that contradicts the solution that Inference Bear has found (but which does not contradict the behavior she wants to specify). Such

Figure 4. Inference Bear’s initial conjecture.

an example is shown in Figure 5. It is identical to the first example besides showing the button at a different position.

Figure 5. The second example.

Inference Bear responds by refining the inference as shown in Figure 6. It has now inferred that the new button position is relative to the original position.

Figure 6. Inference Bear’s refined conjecture.

However, it moves the button by a fixed amount of pixels rather then by one button length. The designer supplies another example (Figure 7) which uses a different button width than the previous examples.

Figure 8 shows the final inference.

1. Keep in mind that the designer does not have to be able to read this specification in order to use Inference Bear - she can also test the inferred behavior interactively. We use the textual version here to communicate the inferred behavior because there is no good way of mapping the interactive testing to paper.

Before

After

Figure 7. The third example.

Figure 8. Inference Bear's final conjecture.

EXAMPLE 2: COLORS

In the simplest case, the demonstrated behavior consists of setting attribute values to constants. A single example suffices to specify this behavior. For example, the designer can demonstrate that clicking on a button changes color of the background window as in Figure 9.

Figure 9. Changing Colors

This is done through two demonstrations. In the first demonstration, the user takes a Before snapshot where the background color is anything but blue, clicks on the "Blue Background" button and takes an After snapshot where the color is blue. The inferred script is shown below.

```
on blue_button.PRESSED() {
        SXWorkArea.background := "blue"
}
```

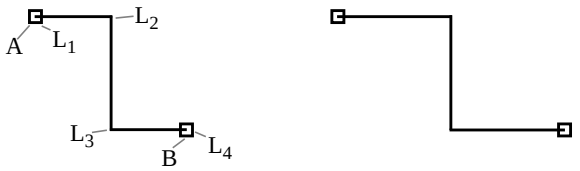
The behavior for the "Green Background" button can be defined in the same way. Alternatively, it is also possible to simply copy the script for the "Blue Background" button in the text editor and to modify it, which avoids having to give a similar demonstration twice.

EXAMPLE 3: ADJUSTING A LINE

The name of "before" and "after" snapshots may suggest that the technique would not be appropriate for describing what happens "in-between". However, this can be addressed by letting the designer control the granularity of the events that behavior is attached to.

Inference Bear lets the user indicate if the demonstrated behavior should be asserted after every move event or after the final move event using a check box.[1] These options are available after clicking the "More" button of Inference Bear's main control panel (Figure 2).

Let us define how a connections behave when the elements they are connected to are moved. This can be done using two examples as shown in Figure 10. The triggering event is a move event of B.



First Before Snapshot *First After Snapshot*

Second Before Snapshot *Second After Snapshot*

Figure 10. Continuous Resizing Example

The inferred script is shown below. This example is simplified - it assumes that elements A and B have negligible height and width. Inference Bear can also make the line position dependent on the height and width of an object (such as "L_4.y := y + 1/2*B.height") but more than two examples would be needed in that case. Inference Bear needs at least *n* examples to infer an assignment which has *n* variables in its right-hand side (a constant offset counts as one variable). This is because Inference Bear does not use domain knowledge in its inferencing but rather solves sets of equations.

1. The same approach was taken in Lapidary [10].

```
if B.moved(Integer x, Integer y) {
        L2.x := 1/2*A.x + 1/2*x
        L3.x := 1/2*A.x + 1/2*x
        L3.y := y
        L4.x := x
        L4.y := y
}
```

This particular example may suggest that we would be better off using one-way constraints as the basis of Inference Bear because we have only captured what happens when *B* is moved (nothing would happen if *A* is moved). With our event-driven assignments, the same behavior must be demonstrated when A is moved while constraints could describe how the connections resize if either is moved. However, one-way constraints cannot adequately describe other functionality such as incrementing and run-time creation and deletion of elements.

EXAMPLE 4: ALIGNING

The intended behavior in this example is that the left sides of B and C aligned to the left side of A. The demonstration consists of two examples as illustrated in Figure 11.

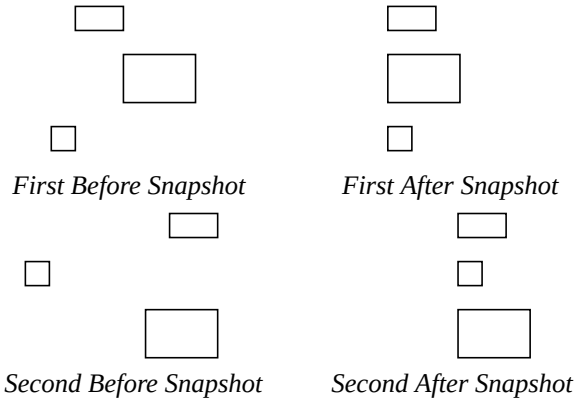


Figure 11. Snapshots for Alignment Example

Here, the source variables are coordinates of the three rectangles. The target values are the horizontal coordinates of B and C.

```
if A.pressed() {
        B.x := A.x
        C.x := A.x
}
```

Inference Bear can also infer aligning left-hand sides (B.x := A.x +A.width-B.width) and aligning centers (B.x := A.x+1/2*A.width-1/2*B.width), which require more examples.

EXAMPLE 5: DRAGGING FROM A PALETTE

Many applications let users create new elements by dragging prototypical elements from palettes. We handle this case by using palettes of actual objects which are in effect copied when they are moved on the canvas.

The demonstration consists of two examples. The first example shows a snapshot of an empty canvas. The trigger event is a drag event of a palette element onto the canvas. The After snapshot shows that a new element *n* has been created at that position and that the palette object *p* returns to its original position which is (20,40) for this example. The second example is identical to the first one besides using a different position for the newly created object.

Inference Bear first finds a prototype by looking for the most similar object for *n* which is *p*. It then computes the attributes for n which are different from p. The only difference is the position here..

```
if p.placed(Integer x, Integer y) {
        object n := p.copy()
        n.x := x
        n.y := y
        p.x := 20
        p.y := 40
}
```

Inference Bear lets users demonstrate many flavors of creating elements by dragging from a palette. The drawback is that the palette object itself is dragged onto the canvas and reappears at its original position only when the mouse is released which can irritate users.

EXAMPLE 6: PARAMETERIZED OBJECT CREATION

This is another example of creating new objects. Clicking on the “New Clock” button *b* creates a new clock object in the canvas. Its initial position and background color are to be taken from input fields *X*, *Y* and *Color* as illustrated in Figure 12. (No claims are made that this is a good interface, of course, we use it for illustration only here.)

Figure 12. Parameterized Object Creation Example

The demonstration consists of two examples which each show that a new object is part of the After snapshot but with a different position and color. The position and color match the contents of the fields.

There is no suitable prototype object for the new object here so that Inference Bear automatically creates one. This is done by permanently copying the new object to the upper left corner of the canvas and making it invisible. This newly created object is named *p* below.

```
if b.pressed()
  object n := p.copy()
  n.visible := true
  n.x := xfield.content
  n.y := yfield.content
  n.background = colorfield.content
```

The visibility attribute is set because the prototype object is invisible in this example as opposed to the prototype object in Example 5.

INFERENCE BEAR ANATOMY

As mentioned earlier, Inference Bear uses the demonstrational engine described in [5]. The most notable characteristic of this engine is that it does not contain domain knowledge. We do not want to explain the inferencing process here but will only describe the user-visible effects of using a domain-independent engine.

The most visible effect is that Inference Bear is "conservative" in its inferences - all its inferences are based on correlating variables in a thorough manner. If there is only a single example, e.g. the one in Figure 3, the system will not attempt to guess a relationship to other variables. It will rather choose the simplest possible solution which solves the demonstration, (an assignment from a constant). Only when the second example makes this solution invalid (Figure 5)does Inference Bear search for a more complex relationship.

A related effect is that users have to give more examples than they would have to if they were using a rule-based system. These systems can usually guess relationships from a single example (assuming, of course, that the system has a rule for e.g. moving a button one button length to its right, and that it fires correctly). However, we feel that the inconvenience of sometimes giving more examples is normally offset by the higher generality of the inference engine.

Another effect introduced by using the inference engine of [5] is that a variable has to be changed in a demonstration before the inference engine attempts to use it in its inferences. This is exemplified in the inference of Figure 6 - the width of the button has never been changed at this point. It takes an example with a different button width to make the intended inference because the inference engine's vision is based on motion. (Another demonstrational system bases its vision on proximity [8].) This is often puzzling to novice users. However, once they learn how to use the inference mechanism they can use it in a wide range of situations.

USABILITY TESTING

Observing actual users is the lithmus test of any demonstrational system. We have informally tested Inference Bear in usability studies. Subjects were given the task of designing examples similar to the ones presented in this paper. They were not given any instructions or assistance other than starting up Inference Bear for them. Most subjects were members of the Graphics, Visualization and Usability Center so that they represent a technical audience. No coherent testing has been done on non-technical subjects. There were a dozen subjects which used the system for about an hour and about five dozen which have used it for about ten minutes.

The testing was not thorough enough to draw statistically significant conclusions but we will present some observations.

- The first observation is that demonstrational systems are potentially, but in no way inherently, easy to use. It took many, often seemingly small, refinements to the user interface until we achieved the performance described below.
- It takes about fifteen minutes to complete the task that specifies how the two buttons change the window color. (More of this time is spent figuring out how to use the interface builder rather than the demonstrational methodology.)
- Users had surprisingly little trouble with demonstrations involving a single example. Demonstrations involving the creation or deletion objects do not seem to be more difficult than others.
- Demonstrations involving two examples seem to be significantly more difficult for users to understand. It takes about fifteen more minutes to do the "moving button" task. All of these fifteen minutes were spend trying to understand how to give the demonstration.
- As explained earlier, Inference Bear eliminates attributes which remain constant across Before snapshots from consideration as parameters. Many subjects were initially puzzled that the Bear would solve their demonstrations using constants (80 pixels) rather than the attribute they meant (the width of the button) when they did not change the width in the examples ("do what I mean!").

LIMITATIONS

Many limitations have already been discussed, such as Inference Bear's current inability to deal with general sets of objects (sets which vary at run time) and the problem of too much user involvement in designating where to attach inferred functionality.

Another problem with Inference Bear, as with most other demonstrational systems, is that users have to edit text directly to combine the functionality of single demonstrations (for non-trivial designs). Thus, they cannot build a complete, substantial user interface design exclusively by demonstration.

CONCLUSION

Inference Bear uses an existing user interface builder and an existing, domain independent inference engine. In this way, we could build it in less than a month, and it contains only about 1500 lines of C++ code.

We feel that Inference Bear shows that it is possible to build an easy-to-use demonstrational system without coupling its inferencing tightly to its domain. Avoiding domain-dependence in this way allows several demonstrational systems to share a common inference engine.

ACKNOWLEDGEMENTS

I would like to thank Siemens for sponsoring part of this research, and Christie Gerlach and Brad Myers for their feedback. The camera, recorder and clock icons were designed by Kevin Mullet of Sun Microsystems.

REFERENCES

[1] Cypher, A., *Eager: Programming Repetitive Tasks by Example*, Proceedings of CHI'91, New Orleans, Louisiana, pp. 33-39.

[2] Fisher, G., D. Busse and D. Wolber, *Adding Rule-Based Reasoning to a Demonstrational Interface Builder*, Proceedings of UIST'92, Monterey, California, November 1992, pp. 89-97.

[3] Foley, J., W. Kim, S. Kovacevic and K. Murray, *Defining Interfaces at a High Level of Abstraction*, IEEE Software, Jan. 1989, pp. 25-32.

[4] Frank, M. and J. Foley, *Model-Based User Interface Design by Example and by Interview*, Proceedings of UIST'93, Atlanta, Georgia, Nov. 1993, pp. 129-137.

[5] Frank, M. and J. Foley, *A Pure Reasoning Engine for Programmning By Demonstration*, Technical Report git-gvu-94-11, Georgia Institute of Technology, Graphics, Visualization and Usability Center, Atlanta, Georgia, Apr. 1994.

[6] Kühme, T. and M. Schneider-Hufschmidt, SX/Tools - An Open Design Environment for Adaptable Multimedia User Interfaces, *Computer Graphics Forum*, 11(3), Sept. 1992, pp. 93-105.

[7] Kurlander, D. and S. Feiner, *Inferring Constraints from Multiple Snapshots*, Technical Report cucs-008-91, Computer Science Department, Columbia University, May 1991 (also to appear in the ACM Transactions On Graphics).

[8] Maulsby, D., I. Witten and K. Kittlitz, *Metamouse: Specifying Graphical Procedures by Example*, Proceedings of Siggraph'89, pp. 127-136.

[9] Myers, B., *Creating User Interfaces By Demonstration*, Academic Press, Boston, 1988.

[10] Myers, B., B. Vander Zanden and R. Dannenberg, *Creating Graphical Interactive Application Objects By Demonstration*, Proceedings of UIST'89, Williamsburg, Virginia, Nov. 1989, pp. 95-104.

[11] Myers, B., R. McDaniel, D. Kosbie, *Marquise: Creating Complete User Interfaces By Demonstration*, Proceedings of INTERCHI'93, Amsterdam, Netherlands, April 1993, pp. 293-300.

[12] Olsen, D. and K. Allan, *Creating Interactive Techniques by Symbolically Solving Geometric Constraints*, Proceedings of UIST'90, Snowbird, Utah, Oct 1990, pp. 102-107.

[13] Singh, G., C. Kok and T. Ngan, *Druid: A System For Demonstrational Rapid User Interface Development*, Proceedings of UIST'90, Snowbird, Utah, Oct. 1990, pp. 167-177.

[14] Wolber, D. and G. Fisher, *A Demonstrational Technique For Developing Interfaces With Dynamically Created Objects*, Proceedings of UIST'91, Hilton Head, South Carolina, November 1991, pages 221-230.