

# PBD and EUP in HabilisDraw

Rob St. Amant

March 4, 2002

We have several motivations for introducing tools into a direct manipulation environment:

- It's a way of working out a theory of tool use, in the context of a specific domain.
- We believe it may lead to new capabilities for users that straightforward direct manipulation may not provide.
- It may give us new insight into programming by demonstration.
- It may give users a stepping stone between direct manipulation and real programming, to provide a type of end user programming.

This brief note concentrates on the two last points. First, what inferences are possible or even necessary in a tool-based system, compared with existing systems for programming by demonstration? Second, what does end user programming look like in a tool-based environment?

## Programming By Demonstration

Consider this example task in a conventional drawing package. The goal is to draw several objects all of the same size, in a straight line. The user draws the first object, then draws the second, and so forth. After the first few objects, the system can infer common properties across the set, including their placement, and can offer to create further objects automatically (given a smart enough system.) The relevant inference made by the PBD system in this case involves the placement of objects in correct alignment. In a conventional system, the alignment action would be made explicitly by the user. This is a simplistic example, but captures a representative kind of reasoning in PBD.

Here's how the user might carry out the task above in HabilisDraw plus a PBD component. The user creates one object and then moves a ruler into place against it. The rest of the objects are drawn automatically aligned, with the user concentrating only on sizing. In some ways this is a more natural way to express the task than creation followed by alignment followed by size adjustment. The placement of the ruler is a substitution for the PBD inference. While this means that the user has to carry out the action explicitly, as in a conventional system, it happens at a different time. The reason that inference is required in the conventional system is that after the objects are created, there's no information about what the user might want to do next, except in the properties of the objects themselves. In HabilisDraw this information can be accessible while object creation is still in progress.

Is this a significant difference? I hope so, and I hope that this idea can be extended to other examples, so that the system can make inferences about the presence of tools that actually reduce the number of actions that the user needs to take.

## End User Programming

Programming is generally associated with three types of control constructs above the level of primitive statements: sequences, iteration, and conditionals. (Primitive statements are the statements in the command language that is currently under development.) Is it possible to create tools that provide these three types of functionality?

As an example, consider a tool for counting or incrementing. Let's treat the value of some property of a tool as a starting state, e.g., the radius of a compass tool. We could imagine attaching a counting tool with a fixed increment to the compass, in order to create a set of concentric circles. This is a specific type of iteration that might be generalized to other repetitive object creation tasks.

Similarly, we might imagine a tool for testing, in the simplest case for equality testing of the value of a tool property. The testing tool would produce two branches, based on the state of the tool on which it acts.

Sequences of operations on tools might be captured by some form of macro recording. The user applies some tools in sequence or edits the properties of some sets of tools, the actions being captured and providing input to some machine learning algorithm.

This is all very preliminary, but I think it is worth looking into. This should not be simply a visual programming environment, in which boxes and arrows substitute directly for iterations, tests, and sequences. Instead, I think it might be possible to create tools with strong physical analogies.