ABSTRACT

HORTON, THOMAS E. HabilisDraw: A Tool-based Direct Manipulation Software Environment. (Under the direction of Associate Professor Robert St. Amant).

Direct manipulation interfaces already employ a weak analogy to the use of physical tools in the real world. Despite certain tradeoffs, a stronger application of tool-using principles can lead to improvements in the design of software interfaces. I outline here some of the theory behind such an approach, and describe the design of systems that follow these principles, with emphasis on a tool-based drawing application called HabilisDraw.

HabilisDraw:
A Tool-based Direct Manipulation
Software Environment

by

Thomas E. Horton

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2004

Approved By:

Dr. Dennis Bahler	Dr. Michael Young
Dr. Robert St. Amant Chair of Advisory Committee	

To my grandparents, Gene and Mary Biddix.

Biography

Born in April of 1979, I grew up in Grifton, a small town in eastern North Carolina. At a young age, the town's low lying area instilled in me a certain fondness of swamps and the murky creeks common to the region. Kindergarten through eighth grade were spent, for the most part uneventfully, at Grifton School. At various times, I tried baseball, orchestra, and band, neatly failing to excel in any of them. As a Cub, and later, a Boy Scout, I came to develop an intense aversion to the perils of leadership, as well as an affinity for getting lost in the woods.

For my first two years of high school, I attended Ayden-Grifton High, where, in March of 1994, I pulled my first all-nighter, in order to finish a report on biological contaminants. It was also at Ayden-Grifton that I learned to speak Latin with a Texan accent, in a class taught by the school's French teacher (believe it or not, this did actually help on the SATs).

In the summer of 1995, I went to a scuba-diving camp on the coast. During our final dive, I miscalculated my own buoyancy, with the result that I spent the entire time upside down. I also managed to run out of air while still twenty feet underwater, underscoring a certain chronic difficulty with meeting deadlines.

After recovering from the record setting four simultaneous ear infections I developed post-scuba camp (despite the obvious handicap of having only two ears), I transferred to the North Carolina School of Science and Mathematics. At NCSSM, I learned how to go for days without eating, showering, or sleeping (which would leave me at a bit of a loss for something to do when I hit freshman year of college). It was here that I was introduced to both computer science and evolutionary biology, which soon became my two favorite subjects.

Graduating with only mild emotional scarring in 1997, I then went to NC State University, where I managed to graduate with a major in computer science (despite never learning to type) and a minor in anthropology (physical anthropology being about as close to evolutionary biology as I could get at an engineering school). Looking out at the real world, I found it somewhat lacking and decided to stay in academia for a while longer. As a grad student, I was quite fortunate to be recruited onto a project studying tool-use from a computer science perspective, unexpectedly giving me a chance to actually put the time I had spent on my minor to use. This thesis is one result of that research.

Acknowledgements

This effort was supported by the National Science Foundation under award 0083281. The information in this paper does not necessarily reflect the position or policies of the U.S. government, and no official endorsement should be inferred.

Contents

List of Figures

Chapter 1

Introduction

The use of tools is one of the most important and universal of all human behaviors. As a species, *Homo sapiens* is supremely adapted to the invention, manufacture, and utilization of tools - from a simple stick used to scrape gum from the bottom of a shoe, to the supercomputers used to study the motions of galaxies.

In many ways, direct manipulation software interfaces are designed to exploit our familiarity with manipulating and applying physical tools in the real world, and metaphors based on tool use may be found in all kinds of software interfaces. Despite this ubiquity however, direct manipulation interfaces typically make only superficial use of the tool metaphor, leaving open the question of how a deeper application of concepts learned from a study of tools and tool use may improve interface design.

The focus of this paper will be a prototype drawing application called HabilisDraw[1], which incorporates concepts of tool-use derived from an analysis of interaction with tools in the real world. HabilisDraw was developed under the direction of Robert St. Amant, as part of a larger ongoing investigation into the application of tool-use concepts to the domain of computer science. Much of the theory outlined here originated with or was developed in concert with him and with other members of our research group, including Ergun Bicici, Colin Butler, David Christian, and John Daughtry.

I begin with a description of current direct manipulation interfaces and suggest why we

[1]For the curious, the name comes from the first species of hominid known to have made and used stone tools, *Homo habilis* ("handy man").

might hope for something better. Chapter 2 develops some general principles of tools and tool use, and considers some of the potential problems with tool-based interfaces. Chapter 3 gives an overview of the previous work in the area. A description of the HabilisDraw application and its development is in Chapter 4. An analysis of HabilisDraw follows in Chapter 5. Chapter 6 introduces some related tool-based applications developed after the original HabilisDraw. Chapters 7 and 8 suggest directions for future study and provide a conclusion, respectively.

1.1 Motivation

In the twenty-odd years since its development, the WIMP[2] model has come to dominate interface design. In that time, software users have demonstrated great skill at adapting to WIMP-based interfaces. Clearly, WIMP works. But as Gentner and Nielsen argue [GN96], the success of WIMP interfaces does not mean we should stop experimenting with alternative, or what Beaudouin-Lafon calls "post-WIMP" [BL00], interfaces.

While designers have had the past two decades to refine the WIMP model, evolution has been at work on humanity's tool-using skills for more than two million years. If we can create software interfaces that exploit our existing proficiency with physical tools, we might expect to develop applications that are more efficient, more flexible, easier to learn, and more enjoyable to use. The use of physical tools often leads the user to feel as though a tool is a part of, or an extension of their own body, rather than a separate, external object mediating their actions. Such a sense of engagement could also become a desirable property of software tools.

1.2 Direct manipulation

As a key feature of WIMP design, the direct manipulation of interface elements is used extensively in software applications. The success of direct manipulation is in no small part due to its close approximation of the way humans interact with objects in the physical environment.

[2]WIMP: Windows, Icons, Menus, and Pointers

Shneiderman defines a set of principles that characterize direct manipulation [Shn83]: objects are represented continuously; functionality is invoked through physical actions performed on objects, rather than with a complex syntax; operations are fast, incremental, and reversible, with immediately visible effects; and a layered or spiral approach to learning. Shneiderman goes on to describe several benefits of direct manipulation: novices can learn to use a new system quickly; experts can perform a wide range of tasks efficiently; intermittent users retain knowledge; error messages are rarely needed; users can immediately tell if an action is productive, and if not reverse it; users are more comfortable because the system is comprehensible and reversible; and users gain confidence and mastery because they feel that they are in direct control and that the system's responses are predictable.

Computers are logical devices - they facilitate the manipulation and display of abstract data. Thus, computers and the software they run are often classified as cognitive, as opposed to physical tools. However, the success of direct manipulation interfaces suggests that the application of physical metaphors to the software domain can improve the effectiveness of computers as cognitive tools.

1.3 Direct manipulation interfaces

Despite the accepted advantages of direct manipulation, most current WIMP interfaces make only a superficial use of the technique. Interactions with individual widgets like buttons and menus conform to the principles of direct manipulation, but at the functional level, direct manipulation is often abandoned. Beaudouin-Lafon points out several of these areas where WIMP interfaces can be seen to violate the principles of direct manipulation [BL00]. For example, dialogs boxes violate direct manipulation by obscuring and making other parts of the interface inaccessible, which breaks the principle of continuous representation of objects. WIMP interfaces also often require a complex syntax instead of a simple physical action: users frequently must select one or more objects, select a command from a menu, fill in parameters in a dialog box, and click "ok" to accomplish a single task. Beaudouin-Lafon refers to this as "indirect manipulation... through (direct) manipulation of the interface elements." The use of menu commands and most dialog boxes[3] runs counter

[3]Some dialogs, which Beaudouin-Lafon refers to as "inspector windows," provide live updating of either the selected object or a preview sample as controls in the dialog are adjusted, in better keeping with direct manipulation.

to the principle that interface actions should be fast, incremental, and reversible operations with immediate effect. Finally, although the small number of interaction techniques used in WIMP interfaces makes it easy for novices to learn the basics, more advanced techniques, such as shortcuts that use modifier keys in concert with mouse clicks, tend to be concealed and inconsistent across applications, making the transition from novice user to expert difficult.

Breaking from the standard WIMP model gives interface designers the opportunity to apply direct manipulation techniques directly to the functional aspects of a system. This is one of the goals of a tool-based interface design.

Chapter 2

Theory

2.1 Tool use

If we are going to apply principles derived from physical tools to the software domain, a minimum understanding of the nature of tool use is required. Tool use has been studied in a variety of fields[1], but one of the most widely accepted definitions of tool use comes from Beck's studies of cognition in non-human primates [Bec80]:

> "Thus tool use is the external employment of an unattached environmental object to alter more efficiently the form, position or condition of another object, another organism, or the user itself when the user holds or carries the tool during or just prior to use and is responsible for the proper and effective orientation of the tool."

Beck's definition summarizes several key aspects of tool use.

- *Tool use involves direct action* [Vau96]. For example, cracking a nut by striking it with a hammer stone is an example of direct action. However, grasping the nut and dashing it against the stone (now used as an anvil) is an indirect action, and generally not considered an example of tool use.

- *Tool use is a goal-directed behavior* [Ing96]. Intent is a necessary aspect of tool using behavior. Though a desirable effect may sometimes be obtained through incidental

[1] A sample of works from different fields: AI and situated cognition [BABC84, AH97]; HCI and cognitive psychology [Bab03, Hut95]; ecological and experimental psychology [vSv94, WC01, WC03]; animal psychology [Bec80, DRZG97, GI93, Pov00, RBP96, TC97]; evolution of cognition [Dea98, Mit96, Ste03].

or accidental utilization of an object, such an action is not tool use.

- *Tool use is effective* [VL96]. If the behavior could not possibly generate the desired outcome, for example using a hammer to try to cut down a tree, it is not a case of tool use.

- *Tool use amplifies or augments existing behavior.* A stick may extend a user's reach. A hammer increases the force with which a carpenter can strike a nail. A vise gives a machinist a third and more powerful "hand."

To gain a better understanding of how tool-based techniques might be applied to interfaces, it is useful to consider some of the different types of tools and how they behave. For example, physical tools transform objects in the physical environment, while conceptual tools help us to transform abstract knowledge or data. Tools may be classified by design or application domain, but most generally by function. Our research group has created a high level taxonomy [SH02], which divides tools into four functional categories:

- *Tools that produce a persistent effect on materials or the environment.* Such tools are classified as "effective" tools. Physical examples of effective tools include hammers, saws, chisels, and paint brushes. In the sense that it may have a permanent effect on a data set, a word processor or spreadsheet application could also be assigned to this category.

- *Tools that provide information about materials or the environment.* Tools of this sort have been called "instruments" in tool use literature [Hut95]. Microscopes, rulers, magnifying glasses, and visualization software fall into this category. Sometimes, an effective tool may have an instrument built-in, as with a power drill with a level attached, a table saw with a scale to indicate the angle of the blade, or a word processor which displays the text that it edits.

- *Tools that stabilize or constrain materials, other tools, or the environment.* These are "compensating" tools. Examples include clamps, vices, straight edges, and miter boxes. Sometimes, an effective tool is by its nature inherently self-compensating. As it cuts, a wood saw makes a groove that constrains its wide blade to a linear path, facilitating a straight-line cut. Jigsaws and keyhole saws relax this constraint by having very narrow blades, allowing them to cut curves.

- *Tools that demarcate the environment or materials.* Such "demarcating" tools distinguish similar regions so that they can be treated differently. This category includes carpenter's pencils, push pins, and working surfaces with fixed markings, such as a seamstress's table.

By considering the design and application of tools from each functional category, several general properties may be derived [SH04]. These may be employed as a set of guidelines for the design of a tool-based interface. The degree to which the interface conforms to these guidelines will have a significant effect on the functionality, efficiency, and usability of the design.

- *Object status and manipulability.* Tools are first level objects with which users can interact directly - their physical or logical properties facilitate manipulation. Tools are also persistent objects. A carpenter can put down a hammer, go eat lunch, and then reacquire the hammer back at the work site (absent an obsessively tidy coworker).

- *Affordances.* Tools have perceivable affordances that indicate how they are to be used. For example, the size and shape of a hammer's handle suggest that it can be grasped, while the heft and shape of the head indicate that it is used for striking.

- *Specialized actions.* A tool is designed to be used in a particular manner. This results from a combination of the intended function of the tool and the characteristics of the tool, the user, and the target. For example, hammers require an arcing swing, screwdrivers rotational motion, and saws linear motion, while a spreadsheet accepts numerical input. A screwdriver would be useless if the head were too big for the slot of the screw, if the screw had no slot, if the human arm could not rotate, or if the goal was to turn a nut.

- *Open-loop vs. closed-loop action.* Closed-loop actions involve continuous monitoring of feedback and readjustment, allowing for a high degree of precision, as when a surgeon makes an incision with a scalpel. In some cases however, it may be advantageous to sacrifice precision for power or speed. Open-loop actions follow an initial set-up stage with a relatively uncontrolled motion, as in the swing of a hammer. Sometimes closed-loop actions are used prior to an open-loop action in a preparatory calibration stage, such as when taking a few slow practice swings with a hammer, prior to driving the nail.

- *Effect locality.* In the real world, a tool generally has to come into direct contact at a single region on its target in order to have the proper effect. A hammer must strike squarely with its head to drive a nail straight. The line of bristles on a broom must contact the dirt to sweep. In the case of tools such as air brushes and pressure washers, the contact is mediated, but the effect is still localized to the end of the emitted stream. In a software environment, such constraints are not enforced by the physics of the real world. For example, a dialog box may effect a change in a window in another region of the screen. In such a case, the increased degree of indirection that results from relaxed effect locality may interfere with a user's ability to understand how the application works and to learn to use the software efficiently.

- *Iteration.* Tool use commonly involves an iterative action, often as a consequence of effect locality. For example, only one nail may be driven by a single hammer blow, thus each nail must be visited in turn. Further, it generally requires several repeated strikes to fully drive a single nail. Sometimes actions may have uncertain results, in which case iteration can minimize mistakes by allowing for monitoring and readjustment in between incremental actions.

- *Material consolidation.* It is sometimes possible to avoid the inefficiency of iteration resulting from effect locality by consolidating materials. If several boards need to be cut to the same length, they may be stacked and then all cut at the same time. Similarly, several data files may be modified at once by grouping them with a batch processing command. In addition to reducing the total number of steps needed to reach a goal, consolidation can also increase uniformity and reduce uncertainty since performing actions on the consolidated object eliminates the need for repeated measurements. Of course, a single mistake will affect the entire set of material, making consolidation a risky technique if resources are limited.

- *Variation and duplication.* Certain tasks may be very similar, yet differ slightly in certain key respects. Driving a roofing nail and driving a railroad spike both require hammering, but using a sledge hammer to reshingle your leaky roof would likely do far more harm than good. Thus toolkits often contain many tools that are all variations of a single base type. Screws come in many shapes and sizes, so a workshop should contain multiple sizes of both Phillips and flat-blade screwdrivers. A toolkit may even

contain several identical copies of a tool. When removing the wheel of a bicycle for instance, it is necessary to have two wrenches of the same size, one for each of the nuts on either end of the axle. In a collaborative environment, keeping multiple instances of a tool can increase efficiency by facilitating parallel actions and may be necessary to prevent bottlenecks and deadlocks.

- *Spatial arrangement.* Placing tools in a particular arrangement in the environment prior to beginning a task can greatly increase the efficiency of the work space [Kir95, Vau96]. Experienced tool users often lay out commonly used tools on the assumption that they will probably be needed and should be easily accessible. Similarly, tools that are often used in concert may be grouped together near where they are typically used. Further, instances of commonly used tools may be kept at multiple locations, reducing retrieval time because a copy is always nearby.

- *Adjustability and composability.* Many tools are made up of components that can be rearranged. Further, tools can often be combined with other tools or materials to enhance or modify their effects. For example, if a bolt is too tight to loosen with a wrench, you might try hitting the end of the handle with a hammer, putting a long pipe over the handle to increase leverage, or using a spray can of lubricant to grease the threads. A hammer, designed to drive nails, can be used with a chisel to carve wood. Multiple command line utilities can be piped together to combine their effects.

- *Opportunistic use.* Tool use may be opportunistic. Tools may be used for purposes other than that for which they were designed, or objects that were not designed to be tools may be substituted if the appropriate tool is not available. If no hammer is within reach, a large flat wrench might do. Or if not even a wrench is handy, a graspable rock with a flat side could be substituted.

2.2 Problems with tool-based designs

While there are numerous potential advantages to a tool-based design, it is important to recognize possible pitfalls and tradeoffs.

One of the advantages of traditional WIMP interfaces is the consistency they provide, both within a particular interface and across applications. Once a user learns how to operate

the basic WIMP interface with its buttons, menus, sliders, and dialogs, he or she can bring to bear a wealth of knowledge gained from experience with past interfaces when confronted with a new application.

Since the interaction with an object in a tool-based environment often varies greatly with the function of the object, users must learn how to use each new tool. In HabilisDraw for example, rulers and pens are used very differently because their effects on the environment are very different. This lack of consistency can be offset by providing visible affordances, by utilizing metaphors that draw from the user's past experience, and by designing the tools to behave as users' intuitions suggest they should. Gentner and Nielsen argue that the rich and fine-grained appearance of real-world objects allow for a wide variation in appearance without impeding recognition, and that similarly rich and fine-grained representations in software interfaces can reduce the need for consistency [GN96].

Another problem area is efficiency. Often, tools can be as fast or even faster than their WIMP counterparts, especially for novice users. For example, the ruler used for alignment in HabilisDraw eliminates the need to select each object to be aligned, as is required in many traditional drawing applications. Sometimes however, the use of a tool may lead to reduced efficiency. Many interface elements in traditional interfaces may not be very intuitive, but have been refined over the years to be very efficient, and are so widespread these days that most everyone knows how to use them. Expert users, who are accustomed to using all the various shortcuts to be found in a given WIMP interface, may be particularly frustrated by the constraints of a tool-based metaphor.

Taken to an extreme, mimicking the physical environment can be absurdly counter productive [Smi87]. As useful as such a function would be, the real world has no "undo" option, but no one would seriously argue that the use of a physical world metaphor should prohibit the inclusion of such functionality in a text editor.

It is important to consider whether adhering to a physical analogy provides increased functionality or learnability, or whether it simply adds an unnecessary constraint on the design. As an example, Apple Computer once made a relatively minor, yet widely derided modification to the volume control widget of their Quicktime ™ Player application, replacing the standard slider widget with a dial. While the dial mimicked the volume controls of most stereo systems, and thus might have been more recognizable to a novice user, the slider is by now so familiar to nearly every computer user that any benefit was far outweighed by the awkwardness of controlling a tiny dial widget with a mouse pointer.

Chapter 3

Previous Work

While many of the techniques implemented in HabilisDraw and its kin are novel, the basic idea of a tool-based interface is not a new one - even conventional commercial software environments incorporate some aspects of physical tool use. An effective use of space is demonstrated in the way that many standard interface elements, including menus, palettes, and button bars rely on grouping similar objects together. Material consolidation can be used to describe the process of selecting multiple objects before selecting a function (such as when selecting multiple file icons and then dragging them to a new directory). Viewed as abstract tools, Unix command line utilities are composable via pipes and adjustable via flags. The concept of affordance is widely applied in direct manipulation interfaces [Gav91, Nor91], though with some debate over the precise usage of the term [Nor99]. Generally speaking however, the resemblance of interface "tools" to true tools conforming to the properties outlined in Chapter 2 is rather remote.

Despite a lack of application to commercial software, there has been a fair amount of research conducted involving tool-based interfaces. Quite often however, tool use was not the primary focus of these investigations, but rather a means to a greater end.

3.1 Toolglasses and Magic Lenses

In 1993, Bier et.al. introduced two new direct manipulation mechanisms as part of a "see-through" interface: Toolglasses, which are effective tools, and Magic Lenses, which act as instruments [BSP+93]. Toolglasses are moveable palettes of widgets that can be positioned over application objects. Taking advantage of bimanual input, the non-dominant hand can control the positioning and sizing of the Toolglass, while the dominant hand selects a widget from the palette using a standard pointer. When the user clicks on a widget, its function is applied to the object underneath it. Magic Lenses act as visual filters, like more powerful versions of standard pixel magnifiers. Positioning a lens over an object might reveal hidden information, modify the display (as in changing a rendering from a filled to a wire-frame representation), or subtract out distracting information. A click through a lens will first have its coordinates modified by the lens before it is applied to the objects underneath.

Toolglasses and Magic Lenses can be composed. Stacked lenses will combine their effects on the regions underneath them. Clicking on a Toolglass widget over a lens will apply the widget's function to the region modified by the lens, while clicking through a lens onto a palette will first modify the mouse coordinates by the lens to determine where on the palette the click should fall. Widgets may be designed to include lenses, and lenses may include widgets.

3.2 Stick tools

In 1996, Raisamo and Räihä proposed a new technique for aligning objects in a direct manipulation drawing environment [RR96]. They considered the three most common alignment techniques in existing drawing packages: selection followed by an alignment command, gravity-aided dragging, and automated constraint-based alignment. It was their conclusion that such techniques tended to be either too imprecise or too complex for novice users, and that all lacked a sense of direct engagement. While this might not be a problem for expert users, Raisamo and Räihä note that studies of CAD programs showed that productivity often suffered because powerful yet complex functions tended to be poorly utilized - users were following the habits they carried over from manual drawing techniques. Thus Raisamo and Räihä's goal was to create tools that provided a better fit to the practices already in

use.

Their solution was a tool they dubbed an "alignment stick". The alignment stick is a simple straight edge in the drawing environment that can be used to push around objects, aligning them against its edge just as one might align objects on a desktop against a ruler. The alignment stick can also make use of bimanual interaction with a mouse in the dominant hand controlling the position and activation of the tool, and a trackball in the non-dominant hand used to adjust the length or rotation of the stick.

The stick-tool concept was later extended [Rai99] to include sticks for shaping, cutting, resizing, and rotating objects. Generally, users both novice and expert, were able to learn to use the stick-tools quickly and effectively.

3.3 KidPad

KidPad [BHD+96] is probably the best known example of a tool-based interface. Incorporating a zoomable work surface and allowing collaborative interaction by way of a dual mouse setup, KidPad provides a flexible environment designed to let children create simple illustrated stories and games. Kids can draw anywhere on the work surface, using tools resembling crayons of different colors (an example of tool variation). Other tools let kids pan and zoom the surface, place text, fill shapes, add hyperlinks, and perform other functions. To avoid fights over tools when kids are collaborating, a special cloning tool is provided which can create a copy of any other tool, as well as copying drawn objects.

Earlier versions of KidPad treated tools much like HabilisDraw does. These "LocalTools" could be placed anywhere on the work surface and retrieved later. LocalTools were top level objects, just like real world tools, with the corresponding opportunities for making effective use of the work space. However, later versions of KidPad have altered the original LocalTools design, presumably for the sake of simplicity (KidPad, like HabilisDraw relies on mouse control of tools, and using mouse buttons to both pick up and drop as well as activate tools can be problematic). Rather than being able to drop a tool anywhere, one tool can now only be swapped for another, after which, the first tool is automatically returned to the tool palette.

Some of the tools in KidPad have different effects when used in collaboration by two users, allowing for a degree of composabilty [TH01]. For example, combining two crayon

tools results in a filled shape with a blended color, while dragging with two hand tools, which normally pan, instead zooms the workspace. X-ray windows are similar to Magic Lenses - drawing through an x-ray window creates lines that can only be seen when viewed through an x-ray window.

3.4 Alternate Reality Kit

Though not focused on tool use, Smith's Alternate Reality Kit (ARK) [Smi87] made extensive use of interaction with interface elements that mimicked physical objects. Input was limited to a single mouse and keyboard.

ARK is a physical simulation environment. Objects have velocity and inertia and can be thrown about the work space. A hand is used to position and activate objects, and a "warehouse" provides a palette for introducing new objects into the environment. Buttons can be associated with objects with a dropping action. Clicking on a button sends the button's message (which may be modified by changing parameters) to the associated object. "Interactors" act like switches for controlling the parameters of the environment (such as gravity and contact collisions). Sliders are used to specify numerical values.

As part of his analysis of ARK, Smith describes a tradeoff between what he called "literal" and "magic" features. Literal features are those that fit the interface metaphor - in this case, those features that behaved like physical objects. Magical features violate the interface's metaphor in exchange for increased functionality. Literalism imposes limitations on the interface, but enhances the learnability of the system. Magic may obscure functionality, but can lead to more powerful capabilities. If the literal aspects of an interface adhere to a metaphor with which users are already familiar, their usage is often immediately obvious. In ARK, the time needed to learn the basics could typically be measured in seconds. Though observations indicated that even novice users were not hampered by a small amount of magic, Smith found that adding a new magic element is relatively expensive, with each new magic feature requiring extra time to learn.

Smith also considered external factors that were neither literal nor magic, such as input devices and system performance limitations, that can force a violation of the interface metaphor without enhancing functionality. For example, the use of an indirect input device like a mouse breaks the physical object metaphor of ARK. Further, button objects in ARK

are positioned by a drag operation and activated with a mouse click. Different mouse buttons are needed to active a button and initiate a drag. This is very similar to the interaction with objects in early versions of HabilisDraw. In both cases, users tended to forget which button was assigned to which action. Smith's observations of ARK users led him to conclude that such external factors were the most troublesome aspects of the interface.

Chapter 4

HabilisDraw

HabilisDraw is a drawing application originally developed as a proof-of-concept for tool-based direct manipulation interfaces. We chose to start with a drawing application because drawing is a familiar domain for most computer users, has a well established set of interaction mechanisms, and the task of drawing on a computer has clear parallels to the use of physical drawing tools in the real world. This section begins with a description of the HabilisDraw interface and a discussion of the HabilisDraw tool set. An overview of some of the decisions made during the design process follows, and an analysis of the system may be found in the next chapter.

4.1 Interaction in HabilisDraw

Most existing software drawing and painting applications already exploit a tool-use metaphor, making use of pens, brushes, "rectangle tools," and so on. Some of these tools, such as those for drawing circles and rectangles via bounding boxes, have no analogs in the real world. Others, such as pens and brushes, do bear some resemblance to their real world counterparts, but employ mechanisms that are often far removed from those of physical tools. By the relatively strict definition of tools we have adopted, none of these traditional interface elements really qualify as tools.

In contrast, the tools in HabilisDraw were designed to mimic physical tools as closely as

seemed practical. The most basic difference from the "tools" in typical drawing applications is that the tools in HabilisDraw are first-class artifacts - HabilisDraw tools are persistent objects rather than transient modes. Tools remain in place on the work surface until the user chooses to move or delete them, and multiple instances of a tool class may exist simultaneously, each instance with its own customized properties.

The early versions HabilisDraw used the mouse almost exclusively, relying on the keyboard only for text input and deleting objects (with <delete> or <backspace>). Later versions also use a keyboard modifier key to eliminate the need for multiple mouse buttons found in the early versions (to support disabled users, the modifier key could be made sticky). The general form for interaction with a tool in HabilisDraw is as follows:

- *left-mouse-down -> drag -> left-mouse-up*: moves a tool to a new location or resizes/rotates the tool
- *left-click*: "grasps" the tool, binding it to the mouse pointer
- *<ctrl> + left-click*: drops a grasped tool back onto the work surface
- *left-mouse-down -> drag*: while a tool is grasped, activates the tool (e.g. causes a pen to draw or a ruler to push objects). Releasing the left mouse button deactivates the tool.

A hold-over from earlier versions provides a second procedure that often saves time for expert users. Pressing and holding the right mouse button while over a tool causes the tool to be picked up, releasing the button drops the tool back onto the work surface. If the left mouse button is pressed while the right button is still depressed, the tool activates just as it normally would. In this way, a tool can be used for a quick task, without the need for separate explicit grasp and release actions.

While only one tool may be grasped at a time, it is often inconvenient to have to put down a pen tool in the middle of a task in order to make a small adjustment to another tool or a drawn object and then have to go back and reacquire the pen. To alleviate this problem, pressing the <ctrl> key while a pen is grasped will take the pen "offline"allowing the user to click and drag on other objects to reposition them or adjust their size and rotation. This is analogous to a common real world behavior, in which a grasped pen or pencil is shifted towards the base of the fingers, freeing the finger tips (it's difficult to describe - just try writing something and then doing most anything else without putting down the pen).

Certain tools, such as rulers and compasses, have drag handles on their surfaces. Clicking and dragging on one of these handles allows the user to quickly adjust the length and rotation of the tool. When a drawn object is selected, drag handles are shown at each of the four corners of the object's bounding box. Dragging these handles rotates the object as in many standard drawing applications. It is also possible to open up a modeless control panel attached to an object via a menu selection. Like the screw on an adjustable wrench, or the switches and dials on many power tools, the control panel gives the user precise feedback and control over all of the adjustable properties of the currently selected tool or drawn object (e.g. length, angle, and color values).

In addition to toggling an object's control panel, menu selections are also used to activate a few other functions that would have been impractical to embody as tool objects. Quitting the application, clearing the screen, deleting and ungrouping objects, and toggling views are all accessed from the menu bar.

4.2 HabilisDraw Toolkit

Some tools in HabilisDraw are already familiar to users of software drawing applications, though the interaction with these tools tends to be very different in HabilisDraw. Pen tools draw lines, and ink wells (similar to the "paint bucket" in most drawing packages) change the color of drawn objects. Other tools, such as rulers (which act as straight edges) and compass tools (for drawing circles), will be unfamiliar to new users. The complete tool kit from the original version of HabilisDraw is shown in Figure 4.1. It includes pens (1), ink wells (2), rulers (3), compasses (4), push pins (5), lenses (6), glue bottles (7), and notation pencils (8).

- *Pens* (fig. 4.2). The primary effective tools in HabilisDraw are pens. Each instance of a pen has an RGB color and a line width associated with it. The color of a pen's tip indicates the drawing color, while the line width is represented by a number drawn on the side of the pen. When used alone, a pen simply produces a freehand line object. More complex drawing actions are possible by constraining the pen's motion with a ruler, which acts as a straight edge, and with a compass, which constrains the pen to the compass's radius.

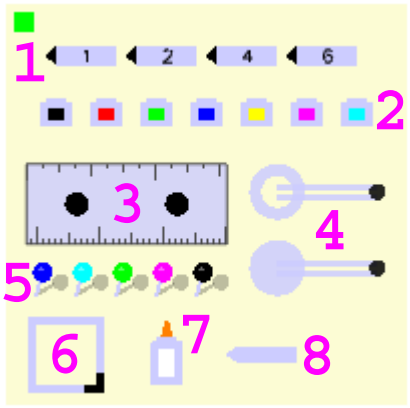

Figure 4.1: HabilisDraw toolkit

Figure 4.2: Freehand drawing with a pen

- *Rulers* (fig. 4.3). Rulers in HabilisDraw have variable length and orientation, which may be adjusted by dragging a ruler's handles. While they can be used as instruments for measuring and comparing lengths, their primary function is as constraint tools by acting as straight edges. A ruler resting on the work surface will constrain the motion of an active pen, forcing it to follow a straight line for as long as the pen is held against the ruler's edge. By arranging multiple rulers, a user can form jigs for drawing lines with specific angles. An activated ruler will push along drawn objects with which it comes in contact, and may thus be used to align objects to the ruler's edge.

  Since a grasped pen will not be constrained to a ruler unless it is drawing, it can be difficult to get the pen exactly on the ruler's edge before starting a line, resulting in lines with short "tails." To avoid this, when a pen tool is brought near a ruler's edge, the edge will highlight - if drawing is then initiated, the pen will "snap to" the edge of the ruler (a magic feature by Smith's definition [Smi87]).

- *Compasses* (fig. 4.4). A compass tool consists of a central base and a handle at the end of a slotted arm. The length and rotation of the arm is adjusted by dragging the

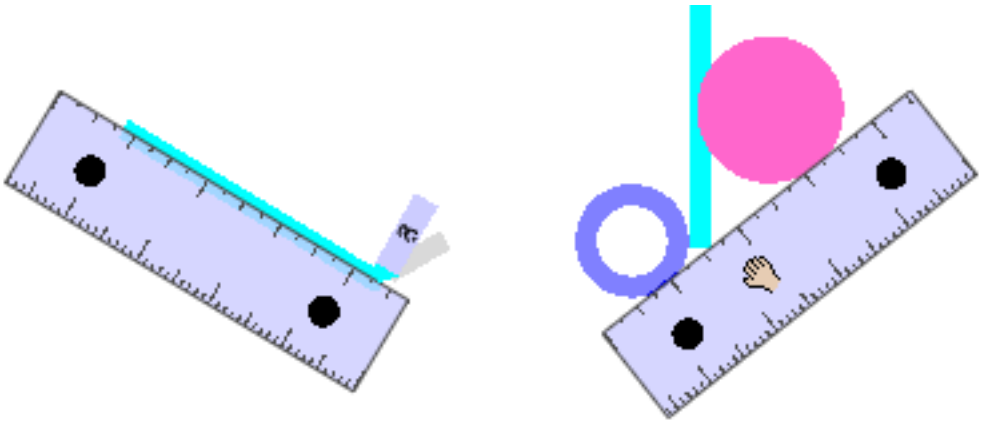

Figure 4.3: Rulers as straight edges

handle. Activating a pen tool while its tip is over the handle of a compass constrains the pen tool to follow the arm as it rotates around the base. Thus a compass tool can be used to draw not only circles, but arcs as well. The pen may move along the slot in the arm, but not beyond the arm's radius. When a pen already constrained by a compass contacts the edge of a ruler, all three tools work together, resulting in an arc with a flat portion where the pen was constrained by the ruler.

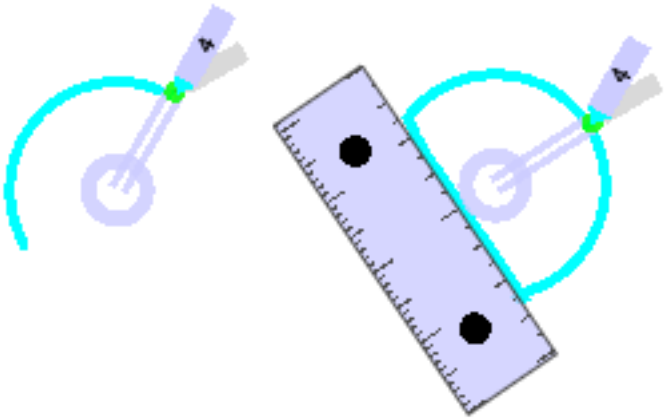

Figure 4.4: Drawing and arc with a compass(left); Combining a compass and a ruler(right)

- *Ink wells* (fig. 4.5). The primary function of an ink well is to simply store a color, indicated by the color of the label on the well. Activating a pen tool while over a well

causes the pen to adopt the color of the "ink" stored in the well. Wells may also be used as effective tools - bringing a grasped ink well's "spout" to a drawn object and activating the well copies the ink color onto the object. Similarly, a well's color may be copied into a pen or another well by activating the well while it is over the other tool.

Figure 4.5: Assigning color with an ink well

- *Push pins* (fig. 4.6). Push pins constrain drawn objects and other tools. A drawn object with a push pin placed on it cannot be moved until the pen is removed. A ruler being pushed along the work surface or a compass being rotated with a pen will stop when it collides with a push pin.

Figure 4.6: Constraining a drawn object with a push pin

- *Notation pencils* (fig. 4.7). Notation pencils are used just like pen tools, and are constrained by rulers and compasses in the same way as pens. Notation pencils are used like carpenter's pencils - the marks they leave are used to draw guides or make notes that are not part of the finished product. Notation pencils draw dashed lines of fixed width to differentiate the marks from drawn objects. The display of these lines can be toggled on and off with a global setting.

- *Glue bottles* (fig. 4.8). When grasped, a single mouse click will place a drop of glue on the surface, while a drag action will leave a trail of glue behind on the work surface (this trail disappears when the bottle is deactivated, like white glue as it dries). If the glue touches an intersection between two or more drawn objects, the objects are highlighted and grouped together. The grouped objects may then be manipulated as a single object. The objects may be ungrouped with a menu command.

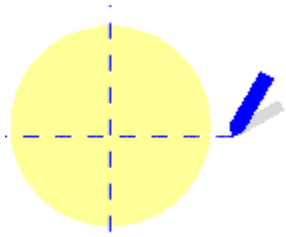

Figure 4.7: Demarcation with a notation pencil

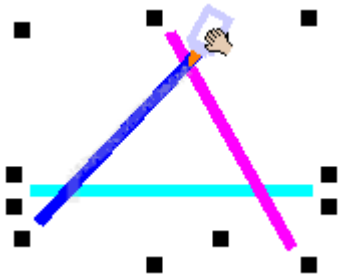

Figure 4.8: Grouping with a glue bottle

- *Magnifying lenses* (fig. 4.9). Magnifying lenses provide a zoomed in view of the work surface underneath. Zoomed tools and drawn objects may be manipulated "through" a lens. The zoom factor of a lens is set via the control panel, and multiple lenses may be overlaid to combine their effects.

Figure 4.9: Using a lens to magnify

Though not a tool itself, there is one other important system object in HabilisDraw. The toolbox contains instances of all the tools in HabilisDraw, much like the palettes in traditional applications. For convenience, some tools, like ink wells, have multiple instances in the toolbox, each with a commonly used set of properties (e.g. primary colors of ink wells, common line widths of pen tools). To conserve space, the toolbox can be collapsed as

well as repositioned anywhere on the work surface. When expanded, clicking and dragging on a tool will tear-off a new clone of the tool. Dropping a tool onto the toolbox removes it from the workspace. Though the current implementation only has a single toolbox with a default set of tools (top-left of fig. 4.10), it would be a relatively simple matter to add support for user-created toolboxes with custom tool sets.

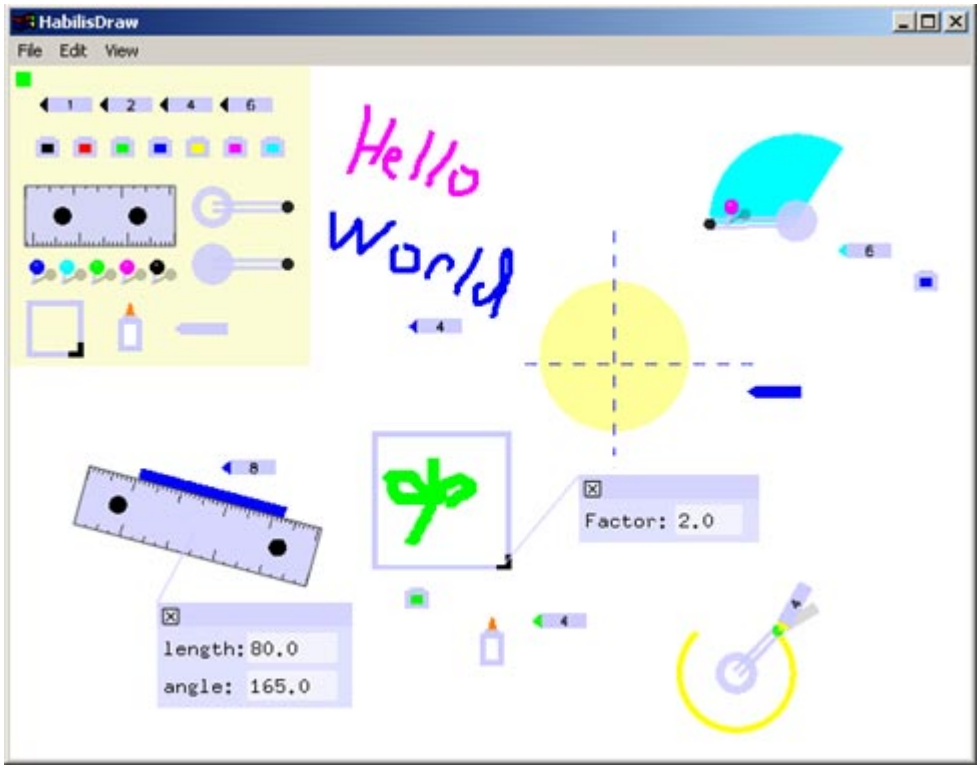

Figure 4.10: HabilisDraw in use

4.3 Design decisions

This section details some of the decisions and trade-offs that were made in the course of developing HabilisDraw.

4.3.1 Mouse limitations

In developing HabilisDraw, one of the requirements was that no special hardware would be needed by the user. This meant that input was limited to a standard keyboard and two button mouse. This poses a problem when considering that the interface requires a large set of actions for manipulating and controlling the tool objects (e.g. move, pick up, put down, activate, resize). Interaction with real-world objects tends to be much richer and more flexible than standard point-and-click mouse actions can easily replicate. Human hands have multiple degrees of freedom and may grasp the same object with a variety of different grips, depending on how the object is to be manipulated. When the user manipulates a ruler tool, he or she is trying to control an object with position, length, and rotation with the single two dimensional point returned by the mouse. While resizing and rotating actions can be combined by dragging one of the ruler's handles, positioning must be performed as a separate action.

On the screen, a compass tool constrains drawing actions to a fixed radius. To translate this constraint into the physical interface, we might have the user control the rotation of the compass with a wheel or crank, providing a high degree of integration [BL00]. No such constraint is possible with the mouse, which is free to move in any direction on the 2D plane. Despite the lack of physical constraint however, users have had no trouble drawing circles with the mouse, with only the logical constraint provided by the compass object.

All objects in HabilisDraw can be picked up and moved. Several can also be resized or rotated by clicking on designated regions of the object. Since movement and adjustment actions do not overlap they can easily be carried out with the same sequence of mouse actions: position the cursor over the body of the object or a handle; click; drag; release. There is however, overlap between positioning and activation actions (e.g. while moving the pen, activating it initiates drawing, deactivating terminates drawing). The most natural way to activate a tool is with a mouse click. In a typical drawing application, to draw a line, the user would click on the line button to set the mode, then click and drag on the work surface to draw. Ideally, in HabilisDraw, a user would click to pick up a pen tool, click and drag to draw, and click to put down the pen. Unfortunately, once the pen was picked up, the system couldn't distinguish between a click to start drawing and a click to drop the pen. Early versions of HabilisDraw used a right-click to pick up or drop an object and a left-click to activate a held object. As Smith discovered with the Alternate Reality Kit

[Smi87], users, especially novices, often became confused as to which mouse button to use. Since one of the main objectives was to create an interface that could be quickly learned by novice users, the use of two mouse buttons was unacceptably complex.

Another option we considered was the use of a double-click to pick up and drop an object. This method suffers from a similar problem, in that the system cannot differentiate between a single click and the first click of a double-click. If, for example, a pen were held and the user double-clicked to drop the tool, the initial click would cause the pen to make a dot before the second click finished the double-click and the pen was dropped. Adding a pause in which the system waits for a second click before taking any action met with some success, but since some users, especially novices may make relatively slow double-clicks, a pause long enough to accommodate such users adds a noticeable lag to the system's response.

We eventually settled on an approach that uses a keyboard modifier. The first click on an object picks it up. Subsequent clicks activate the object. To drop the object, the user clicks while holding down the <ctrl> key.

In a related project, Colin Butler developed a version of HabilisDraw that overcomes some of the limitations of mouse input by using a DiamondTouch touch-sensitive table, modified to support bimanual interaction (Section 6.1.2). This version allows objects to be moved and oriented in a single action and removes the need for handles, significantly simplifying interaction. By supporting bimanual input and actions much closer to their real world analogs (for example, picking up an object is accomplished by pinching an object between thumb and forefinger and raising the hand off the table), the DiamondTouch provides a much more natural style of interaction. At the same time however, the absolute positioning of the DiamondTouch surface introduces new problems. With mouse input, a tool can be easily constrained to the edge of the ruler or the radius of a compass, even though the mouse may still be moving. Most users are familiar with this sort of behavior, which is similar to the bounding of the cursor by the edges of the screen. With the DiamondTouch however, the user's own finger is used as a cursor, and there is no way to physically constrain the user.

4.3.2 Line drawing

In the case of HabilisDraw, drawing a straight line is more complicated than in typical drawing applications, which generally require little more than a single dragging action. Efficiency has been traded for increased functionality and more intuitive behavior. By relaxing the physical tool metaphor slightly and introducing a little bit of magic behavior, it would be easy to add a second line tool to mimic the traditional behavior, but as of this time, we have chosen to keep the metaphor intact.

4.3.3 Alignment

One area where HabilisDraw differs greatly from traditional drawing applications is alignment. Raisamo gives an overview of the different alignment techniques used in commercial drawing packages [RR96]. The most common technique is to select each of the objects to be aligned and then choose an alignment command from a menu, palette, or dialog. This often leaves open the question of which objects will be moved. Possibilities include using the first or last object selected or the farthest object in a particular direction as a base object, or moving all the objects towards some average position (often the only way to be sure is through trial and error or a search of the documentation). Command-based alignment also restricts the alignment to a fixed set of axes (typically just vertical and horizontal alignment) - there is no way to align objects to an arbitrary angle.

More advanced techniques provide greater flexibility, but tend to make simple alignment tasks more complicated. These include snap-to grids and alignment objects, as well as automatic constraint solvers.

HabilisDraw follows Raisamo's solution pretty closely, aligning objects to the side of a straight edge (a ruler in HabilisDraw, the alignment stick in Raisamo's application), eliminating the confusion about which objects will move and allowing alignment to any angle. This solution is simple, intuitive, and flexible, but not without its own flaws. For example, it becomes difficult to align two objects when there is an intervening object that should not be moved.

The use of traditional alignment commands also makes it easy to align objects by their centers rather than by their edges. Initially, the alignment stick could be switched between center and edge alignment modes. However, Raisamo found that the lack of a real-world

analogy for center alignment combined with a lack of visual cues led users to forget not only how to activate center alignment, but that center alignment was even an option. Raisamo's solution was to make edge vs. center alignment a property of each individual object. At this time, HabilisDraw lacks a center alignment mode, though future versions may implement such a feature, perhaps by specifying the point of alignment on an object with a push pin.

4.3.4 Control panels

Following the arguments of Beaudouin-Lafon, the use of dialogs with checkboxes and text fields to adjust the properties of objects may at first seem to be a violation of the principles of direct manipulation and tool use. However, this can be considered as a special case in which dialogs do conform to a tool-based interface scheme. Many physical tools have elements that can be adjusted to alter their behavior. An adjustable wrench has a screw that can be turned to set the spacing of the jaws. A bit driver has several interchangeable bits for turning different types of fasteners. A ratcheted wrench or screwdriver has a switch to change between tightening and loosening modes. A table saw has a crank to set the height of the blade. All of these properties are set prior to the application of the tool.

Thus, using a dialog to adjust the properties of a tool is analogous to opening a control panel integrated into the tool itself. Since the dialog is modeless, the control panel can be left open, providing dynamic feedback showing the current state of the tool. The dialog acts as a sort of meta-tool, not directly interacting with the environment, but instead acting upon its associated tool. Beaudouin-Lafon classifies such modeless interface elements as "property boxes", to differentiate them from dialog boxes [BL00]. Since they can be left open and activated with a simple pointing action, rather than being activated through a menu selection, property boxes have a lower degree of temporal indirection than dialog boxes.

Early versions of HabilisDraw used a single control panel in a separate window. The control panel was logically, but not visibly, linked to the currently selected object in the main window. Though the dialog was modeless, it still violated the principle of effect locality, since there was no visible connection and often a large distance between the dialog and the tool that it affected. At the same time, the early versions of HabilisDraw also provided feedback via a text display that appeared next to a tool as it was resized or rotated. Later versions combine these two features (fig. 4.11). Opening an object's control

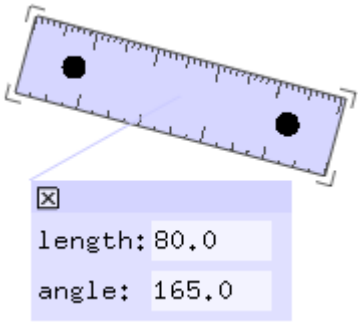

Figure 4.11: Ruler with control panel open

panel now places the dialog next to the object, with a line connecting the two. As the tool moves, the control panel travels with it. In case the dialog is in the user's way, it can be repositioned anywhere on the work surface, but remains visibly and logically tethered to its tool. Integrating the control panel with the object reduces the degree of spatial indirection and also allows multiple control panels to be open simultaneously, facilitating side by side comparisons of settings.

4.3.5 Glue bottles

A common feature of drawing applications that was missing from early versions of HabilisDraw was a mechanism for grouping objects. In typical applications, grouping is accomplished by selecting multiple objects and then choosing a "group" command from a menu. To add a similar function to HabilisDraw, the glue bottle tool was created, using a metaphor familiar to most anyone who played with paper and white glue as a child.

In most drawing applications, objects may be grouped together, even if they do not touch - which can lead to confusion as the lack of visual cues often leaves the user unclear as to which objects are grouped together. In the real world, it is difficult to think of a case where two objects may be rigidly joined without some form of direct contact. In the design of HabilisDraw, the functionality of grouping non-overlapping objects was sacrificed for more intuitive behavior - the glue bottle will only join objects when the glue is placed at a point where the objects overlap.

Grouping can be a relatively complex procedure, involving multiple objects and benefiting from a dedicated tool. Ungrouping however, is a much simpler task, requiring only a

single selection of the composite object and a command. It seemed impractical to create a new tool that did nothing but ungroup objects, and counter-intuitive to overload the glue bottle with an ungroup function, thus the ungroup command remains (for now) as a menu item.

Chapter 5

Analysis of HabilisDraw

This chapter compares the design of HabilisDraw to the principles of tool-based interaction from Chapter 2 and includes a task-based comparison with a typical drawing application.

5.1 Application of principles

HabilisDraw includes examples of tools from each of the classes in our taxonomy. This section examines how these tools are implemented, in terms of the guidelines outlined in Chapter 2.

- *Object status and manipulability.* All the first order tools in HabilisDraw are persistent (though some basic functions such as drawn object rotation and edit and view commands are not encapsulated in top level objects). They may be picked up, carried around, used, dropped anywhere on the work surface, and left for later. When it is functionally appropriate, tools may be rotated and resized. Since interaction is limited to two dimensions and must be mediated by a mouse, rotation and resizing must be performed before a tool is picked up.

  Such a separate manipulation stage is compatible with the way humans use physical tools. Tool use in the real world often requires two distinct phases: an initial setup

phase, in which tools and materials are brought into the required position and orientation, and an action stage, in which the tool is actually applied. As a real world example, a user might first bring a straight edge to the work area, then adjust the angle, and only then use it to draw a straight line. This is exactly how the task is accomplished in HabilisDraw.

- *Affordances.* HabilisDraw tools have affordances that most non-tool-based interfaces designed for the same tasks do not. These are mainly perceived affordances rather than physical affordances [Nor99].

  In some cases, the perceivable affordances of a tool in HabilisDraw are based on an abstract representation of the physical affordances of its real world counterpart. For example, pen tools reproduce the salient visible features of real pens - long and narrow affording a precision grip, with a pointed tip indicating the point of effect. Ruler tools similarly resemble real world rulers.

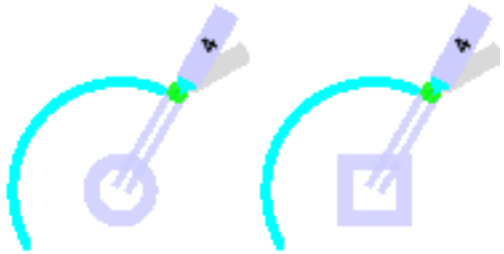

Figure 5.1: Designs for the compass: "good" (left) and "bad" (right)

Other tools, such as compasses, bear little resemblance to real world objects, which are often difficult to represent in a two dimensional interface in a reasonable way. In HabilisDraw, the compass suggests its function by resembling the shapes it creates. Contrast the compass shown on the left in Figure 5.1 with the modified representation on the right - even if these two tools had the same functionality, the visual representation of the actual tool is more consistent with its behavior (though the square "compass" might be a good design for a tool that draws rectangles). The compass tool provides a "true" affordance for drawing circles for two reasons. First, it constrains the movement of the pen to a circular path, and such physical constraints are closely associated with true affordances [Nor99]. Second, this dynamic constraint

is symmetrical with its appearance. This is more than a cultural or logical convention, such as changing the cursor icon to an arc, indicating that a circle will be produced by a dragging action. Using symmetry to predict the task for which a tool is designed is illustrated by the real world example of matching the shape of a screwdriver to the head of a screw.

- *Specialized action.* In a typical drawing application, straight lines, rectangles, and circles all tend to be created in the same manner: the user clicks the icon on the tool palette with the desired shape; the cursor changes appearance; and the user clicks and drags to specify a diagonal of a bounding box. This approach definitely has its strengths - once a user learns how to create one kind of object, they can create any of the other types. But while users can quickly learn how to create objects in these interfaces, it is often very difficult for them to learn how to form the bounding box to generate just the right size, shape, and position.

  In HabilisDraw, the process of drawing a circle is identical to that of drawing an arc (which is often a complicated task in typical drawing programs, but quite simple in HabilisDraw), but very different from drawing a straight line. However, both actions are consistent with the user's experience with drawing circles and lines in the real world, and so can be quite intuitive. Another advantage is that the results of a drawing action can be quite predictable - a circle will be centered on the compass tool, with a radius equal to the length of the compass arm. Further, it can be expected that specializing actions to the class of a tool will improve compatibility between the actual tool and the user's cognitive representation, facilitating recall and increasing familiarity, which should help offset the increased memory requirements of multiple interaction mechanisms.

- *Open and closed loop actions.* Drawing tends to be precision work, so open loop actions in a drawing task (aside from scribbling with a pen) are generally counter productive. Despite this, supporting open loop actions as alternatives to targeted movements, when feasible, does reduce action execution time [DSZ99]. While it is not widely applied in the interface, HabilisDraw does allow for some open loop actions. As a simple example, if the final position of the objects is not a concern, a ruler may be used in an open loop alignment action. As another example, when drawing a circle, there is no difference between an arc that is exactly 360 degrees and one that exceeds

that amount. Thus, once the compass has been set to the proper radius (a closed loop action), all the user needs to do is activate the pen over the compass handle and drag in a vaguely circular motion - multiple identical circles can be created without the need for further closed loop actions.

- *Effect locality.* Non-localized effects run counter to the principle of direct manipulation, increasing the indirection of the interaction, and making the system's behavior less predictable. Traditional drawing applications avoid this by restricting the effects of most user actions to the region around the cursor. Similarly, effects in HabilisDraw are localized around the active tool.

  For certain functions, traditional interfaces must resort to indirect menu commands or global modes. While the activation of these functions follows the rules of direct manipulation - clicking on "OK" button, or selecting an item from a menu for example - the effects these functions cause may be far removed from the dialog or widget used to activate them. In the case of alignment, traditional applications often require a series of localized selection actions, followed by an indirect menu-based function. In HabilisDraw, effects need not be restricted to a single point to be localized. The design of the ruler tool allows for alignment to occur along the entire length of the ruler's edge. Objects on opposite sides of the workspace can be aligned without violating the principle of effect locality because the ruler's geometry extends its effective area beyond the single point of the mouse cursor.

- *Iteration.* In drawing applications, iteration is often limited to the process of creating multiple identical copies of a drawn object. Typically, this is accomplished by selecting the original object, invoking the copy function, and then pasting as many copies as needed. While this violates effect locality (it is often difficult to predict where a new copy will appear), this technique is now ubiquitous in the software world because it is extremely efficient. KidPad [BHD+96] embodies the duplication function with a "clone" tool. Using the clone tool on a drawn object generates a copy just next to the original.

  Though HabilisDraw includes the standard menu-based copy/cut/paste functions, a less efficient, though more direct method is also available. Compasses, rulers, and pins can be combined to form "jigs." A pen tool can then be used in repeated combination

with a jig to create multiple identical objects. While less efficient than cloning, jigs have the advantage that they allow for variation. The shape of a jig can be modified to create altered forms of a basic object. Different pens may be used with the same jig to create variants with the same base shape. A jig may also allow for variation by under-constraining the pen's motion - in the simplest case, a single ruler allows the user to create multiple lines at the same angle, but of different lengths.

- *Material consolidation.* As in most drawing applications, the emphasis in HabilisDraw is on creating new objects from scratch, rather than modifying existing materials. At present, the ruler is the only tool that can operate on multiple objects simultaneously. However, an alternative is to use a glue bottle to group multiple drawn objects into a single composite object. Such a composite object may be rotated or repositioned, recolored with an ink well, or pinned with a push pin.

- *Variation and duplication.* One of HabilisDraw's most basic features is the support for multiple instances of any class of tool, each with its own particular settings. This allows the user to customize the tool set to the particular task at hand.

- *Spatial arrangement.* While new instances of the base tool types in HabilisDraw are always available from the tool box, tools can be left anywhere on the work surface. This allows for many different techniques related to the efficient use of space [Kir95, Vau96]. Tools for one task can be left at the ready in their work area, while the user tends to another task. Tools can be ordered and grouped according to the user's preferences, rather than in a fixed palette. If a task requires switching back and forth between tools, the costs of tool selection and preparation can be reduced by keeping often used tools next to the work area, eliminating the need to swap tools and settings via palettes and dialogs. This can be an even greater advantage if multiple tools of the same type but with different settings are needed - instead of using a color palette to keep swapping the pen color, a pen tool of each color being used can be kept in the work area.

- *Adjustability and composability.* Like tools in the real world, tools in HabilisDraw are often more effective or more efficient when they are used in combination. The simplest case is using an ink well to quickly assign a new drawing color to a pen tool - moving a pen over an ink well and left clicking copies the ink well's color into the pen. This

effect is easily achieved in a conventional drawing application, but the mechanism in HabilisDraw more closely resembles the interaction of plausible real-world tools.

More complex tool interactions are also possible with HabilisDraw. Tools such as the ruler and compass may be used to constrain the motion of a pen tool as it draws. A pen may be used in combination with a single ruler for drawing lines of a given angle, used with a compass for drawing circles and arcs of a given radius, used with several rulers to form corners, or used with both rulers and a compass to generate semi-circles. Using the tools in combination is often easier than trying to draw these objects free-hand, and once the tools are in place, the user can use them as a jig.

- *Opportunistic use.* The opportunistic employment of objects as tools in the real world benefits from the rich set of physical properties inherent in the environment. No one had to say, "hmmm... someone might want to hit something with this rock someday, I'd better make it hard." In an interface, it is difficult to support opportunistic use because it is difficult to anticipate all the possible things a user might try. Making the interface physically consistent is one partial solution. For example, the straight sides of the tool box can be substituted for a ruler when drawing a straight line. Fortunately, the need for opportunistic support is reduced by the constant availability of resources in the interface. For example, the user can never be without a pen, because he or she can always pick up a new instance from the tool box.

5.2 Task-based analysis

While many of the mechanisms in HabilisDraw are more intuitive than their counterparts in traditional applications, there is sometimes a trade-off in efficiency. Drawing a straight line in a typical drawing environment requires nothing more than selecting the appropriate mode and dragging out a line. Broken down into steps, such an action looks something like this:

Move to line icon on tool palette.
Click line tool icon.
Move to line start.
Mouse down.
Move to line end.
Mouse up.

In HabilisDraw however, a ruler tool must first be positioned and used to constrain a pen tool, looking more like this:

Move to one end of ruler.
Mouse down.
Move to line start.
Mouse up.
Move to other end of ruler.
Mouse down.
Move to line end.
Mouse up.
Move to pen tool.
Mouse click.
Move to line start.
Mouse down.
Move to line end.
Mouse up.

As the task becomes more complex however, such as when making multiple similar lines (especially lines with different properties) or making lines with corners, the initial investment in setting up a jig often makes HabilisDraw more efficient than typical drawing applications.

Chapter 6

Other Tool-based Environments

6.1 HabilisDraw Extensions

While the original HabilisDraw has undergone continued refinement, two new offshoots have also been developed. The first introduced a new family of tools to the existing toolkit, while the second completely redesigned the interface to make use of a new input device.

6.1.1 Power and Composite tools

One extension of the original HabilisDraw interface was the addition of two new classes of tools by David Christian and John Daughtry [DS03]. Bar tools allow for more flexible tool composition by creating rigid connecting bars to which other tools may be attached. For example, connecting several pens at regular intervals to a bar creates a new tool for drawing parallel lines like those of a staff in music notation. Bars can also be set to vary their length over time, adding a degree of automation to HabilisDraw. Mover and rotator tools further contribute to automation. When attached to a bar, movers and rotators give a composite tool linear and rotational impetus, respectively. This allows users to easily accomplish new tasks, such as drawing spirals (rotator + pen + shrinking bar - fig. 6.1), that would be next to impossible in any other direct manipulation drawing application.

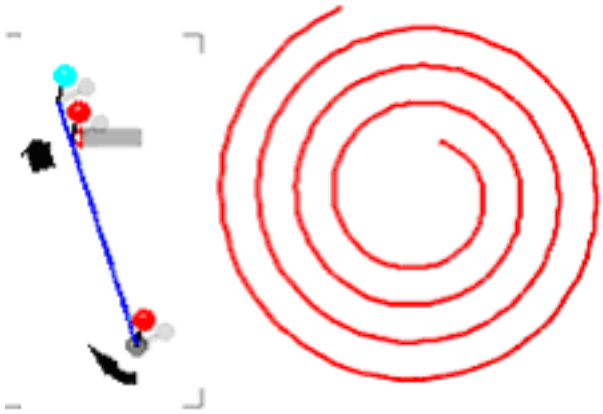

Figure 6.1: Using power and composite tools to create a spiral

6.1.2 HabilisDraw for the DiamondTouch

Colin Butler has performed a complete rewrite of HabilisDraw in order to take advantage of the Mitsubishi DiamondTouch touch sensitive input device [BS04] (fig. 6.2). While pens, rulers, and ink wells remain basically unchanged in HabilisDraw DT, drawing occurs on explicit, manipulable sheets of "paper" taken from an infinite stack. New tools in HabilisDraw DT include a cutting arm and a tape dispenser, used for cutting and joining pieces of paper respectively, and a trash can for removing unwanted tools and materials. Overlapping pieces of paper can act as stencils for drawing. Multiple pieces of paper can be cut at the same time, allowing for material consolidation.

6.2 Smithy

Smithy is an application for solving simple traveling salesman problems, using techniques from human guided simple search (HuGSS) [AAL+99]. In a HuGSS system, the computer uses a simple search procedure to find local minima. Through information visualization, the user identifies promising regions of the search space for the computer to explore and helps the computer break out of non-optimal minima.

Smithy uses a standard simulated annealing algorithm to minimize the path length of the salesman's route. The state of the system is displayed to the user as a connected collection of points within a frame in the application's window (fig. 6.3). The frame's

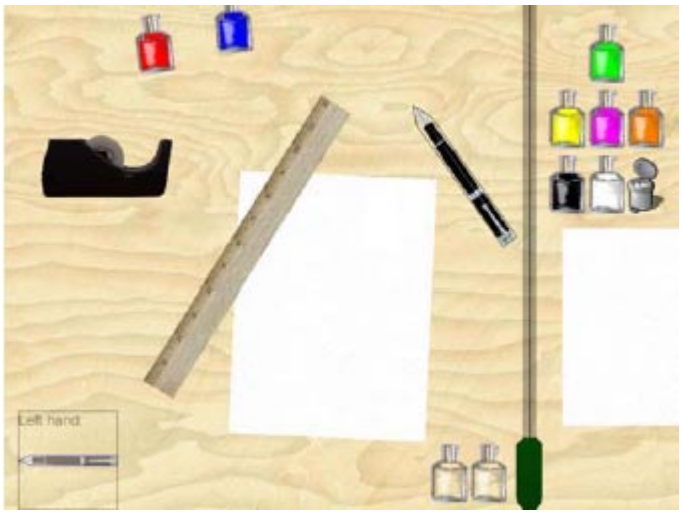

Figure 6.2: The HabilisDraw DT interface

contents represent a map, with each point corresponding to a city the salesman must visit, and the lines connecting the points showing the route the salesman will follow. Initially, the path is simply a random solution to the problem. The user initiates the annealing algorithm by specifying the initial "energy" introduced into the system. A high energy will likely result in a very different solution, as it will cause the algorithm to make modifications to the path that may increase the path length in the short term (thus allowing the algorithm to break free on non-optimal minima). A low initial energy will tend to make local refinements to the existing solution without major global changes. After the algorithm completes, the display is updated with the new solution.

The toolkit in Smithy is much smaller than that of HabilisDraw, comprising a city placement tool, a route highlighter, a hammer, and push pins. Grasping the placement tool and clicking within the frame places a new point on the map. Mousing over a point causes a push pin to appear. The push pin may be dragged within the frame to reposition the city, or dragged outside of the frame to delete the city. Using the highlighter tool on a path linking two points highlights the link and constrains the algorithm by preventing the links removal. Similarly, clicking and dragging from one city to another with the highlighter forces a link between the two points. When the highlighter's tip is positioned over a highlighted link, an "x" appears, and a click will remove the highlight, freeing the link for modification by the annealing algorithm.

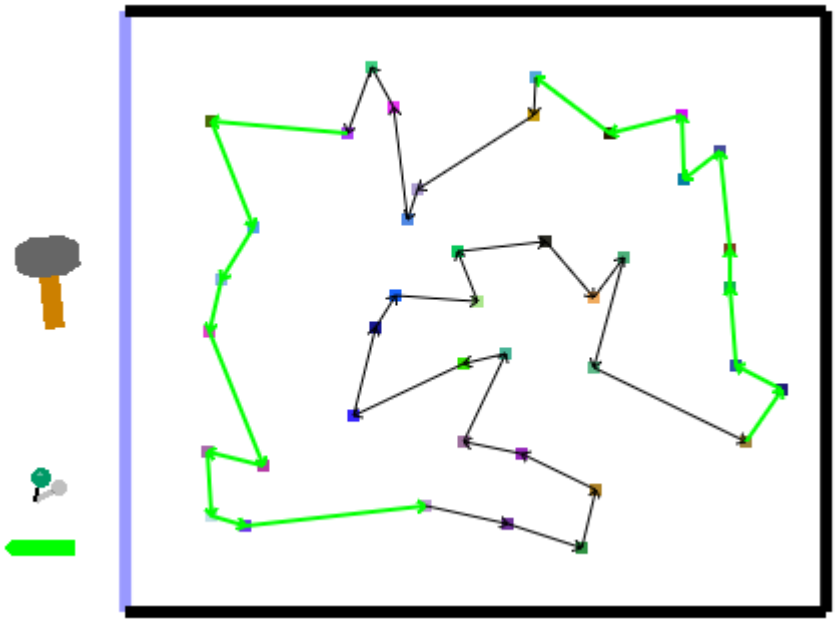

Figure 6.3: The Smithy interface

While it is lying on the work surface, the hammer is drawn horizontally. When grasped, the hammer rotates to a vertical orientation as it is brought closer to the frame - simulating a striking motion. When the hammer connects with the edge of the frame, "energy" is imparted to the system, initiating the annealing algorithm. The magnitude of the energy introduced is linearly proportional to the horizontal component of the hammer's velocity. Thus, small taps with the hammer result in minor refinements to the existing route, while a high-speed impact will cause major perturbations in the solution. The open-loop striking action allows for rapid iteration of the annealing algorithm, without the need to explicitly modify the initial energy input. The analogy to the physical action of striking a container to bring its contents to an optimized state is familiar to anyone who has ever tapped the side of a measuring cup to even out the level of the flour inside. Thus it is easy to guide the search process, even if the user has no knowledge of how a simulated annealing algorithm works.

Chapter 7

Future work

While there are no current plans to further develop HabilisDraw or its kin, there are possibilities for future study.

End-user programming, including the creation of scripts and macros, is common in applications such as spreadsheets and databases - domains with clear symbolic and numerical representations of objects. End-user programming is rare in systems such as drawing applications, where there is a significant conceptual gap between the objects with which the user interacts and the underlying symbolic representation. HabilisDraw and its kin encapsulate numerical and symbolic information in explicit tools, which then mediate user actions. The use of tools provides a possible mechanism for bridging the gap between direct manipulation and programming.

Another possibility is programming by example [Lie01]. In a typical drawing application, the heavy use of modes and dialogs gives few clues to the system as to the intentions of the user. In a tool-based interface, the placement and configuration of the tools lends a degree of structure to the user's activities, providing cues that could be used by an automated assistant to predict and support the user's goals.

Further use of the DiamondTouch in concert with tool-based interfaces also opens up new possibilities, such as the integration of tangible interface techniques. For example, a user might construct a guide tool in the real world that can be overlaid on the DiamondTouch surface. The tool could be used to either physically constrain the user's actions, or have its shape read into the application to create a digital representation of the tool. Other

technologies similar to the DiamondTouch can also detect the user's proximity to the table surface, which could allow the user to perform actions that usually require three dimensions, such as hammering.

Finally, there is the question of how well tool-based interfaces generalize to other domains. A drawing application is an obvious choice for tool-based techniques because real-world drawing depends on tools. But what about more abstract applications, such as spreadsheets? Early evidence seems to suggest that symbolic applications can benefit from tool-based techniques. For example, Robert St. Amant has developed a find and replace dialog for word processing applications that overcomes many of the limitations of traditional find/replace mechanisms, by using elements of tool-based interfaces [SH04].

Chapter 8

Conclusion

Direct manipulation has been one of the greatest advancements in the design of software interfaces, but its application is often limited and superficial. The reliance on modes and dialogs in most WIMP interfaces breaks the direct manipulation metaphor, interfering with the user's sense of engagement and making systems harder to learn and use efficiently.

The addition of explicit tools to the interface provides for a deeper integration of direct manipulation techniques with the functional aspects of a system. This can provide enhanced functionality, as well as make applications easier to learn and use, especially for novice users, while increasing the user's sense of engagement with the system.

There are dangers in this approach however. Over-constraining an interface simply to make it conform to a tool-based metaphor can reduce efficiency and force the designer to remove useful functionality. Careful consideration should be given in deciding whether to use tool-based techniques and in selecting appropriate metaphors.

Bibliography

[AAL+99] David Anderson, Emily Anderson, Neal Lesh, Joe Marks, Ken Perlin, David Ratajczak, and Kathy Ryall. Human-guided simple search: combining information visualization and heuristic search. In *Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation in conjunction with the eighth ACM internation conference on Information and knowledge management*, pages 21–25. ACM Press, 1999.

[AH97] Philip E. Agre and Ian Horswill. Lifeworld analysis. *Journal of Artificial Intelligence Research*, 6:111–145, 1997.

[Bab03] C. Baber. *Cognition and Tool Use: Forms of Engagement in Human and Animal Use of Tools*. Taylor and Francis, 2003.

[BABC84] J. M. Brady, P. E. Agre, D. J. Braunegg, and J. H. Connell. The mechanic's mate. *Artificial Intelligence*, 72(1–2):173–215, 1984.

[Bec80] B. B. Beck. *Animal Tool Behavior: The Use and Manufacture of Tools*. Garland Press, New York, NY, 1980.

[BHD+96] Benjamin B. Bederson, James D. Hollan, Allison Druin, Jason Stewart, David Rogers, and David Proft. Local tools: an alternative to tool palettes. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 169–170. ACM Press, 1996.

[BL00] Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453. ACM Press, 2000.

[BS04] Colin G. Butler and Robert St. Amant. Habilisdraw dt: a bimanual tool-based direct manipulation drawing environment. In *Extended abstracts of the 2004 conference on Human factors and computing systems*, pages 1301–1304. ACM Press, 2004.

[BSP+93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80. ACM Press, 1993.

[Dea98] T. W. Deacon. *The symbolic species: The coevolution of language and the brain.* W. W. Norton & Company, 1998.

[DRZG97] C. Dent-Read and P. Zukow-Goldring, editors. *Evolving explanations of development: Ecological approaches to organism-environment systems.* American Psychological Association, Washington DC, 1997.

[DS03] John M. Daughtry and Robert St. Amant. Power tools and composite tools: integrating automation with direct manipulation. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 233–235. ACM Press, 2003.

[DSZ99] M. S. Dulberg, R. St. Amant, and L. Zettlemoyer. An imprecise mouse gesture for the fast activation of controls. In *INTERACT '99*, pages 375–382, 1999.

[Gav91] William W. Gaver. Technology affordances. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 79–84. ACM Press, 1991.

[GI93] K. R. Gibson and T. Ingold, editors. *Tools, language, and cognition in human evolution.* Cambridge University Press, Cambridge, UK, 1993.

[GN96] Don Gentner and Jakob Nielsen. The anti-mac interface. *Commun. ACM*, 39(8):70–82, 1996.

[Hut95] E. Hutchins. *Cognition in the Wild.* MIT Press, Cambridge, MA, 1995.

[Ing96] E. J. Ingmanson. Tool-using behavior in wild *pan paniscus*: Social and ecological considerations. In A. E. Russon, K. A. Bard, and S. T. Parker, editors, *Reaching into Thought: The minds of the great apes*, pages 190–210. Cambridge University Press, Cambridge, UK, 1996.

[Kir95] D. Kirsh. The intelligent use of space. *Artificial Intelligence*, 73:31–68, 1995.

[Lie01] H. Lieberman. *Your Wish is My Command: Giving Users the Power to Instruct their Software*. Morgan Kaufmann, 2001.

[Mit96] S. Mithin. *The Prehistory of the Mind: The Cognitive Origins of Art, Religion, and Science*. Thames & Hudson, London, 1996.

[Nor91] D. A. Norman. Cognitive artifacts. In J. M. Carroll, editor, *Designing interaction: psychology at the human-computer interface*, pages 17–38. Cambridge University Press, Cambridge, UK, 1991.

[Nor99] Donald A. Norman. Affordance, conventions, and design. *interactions*, 6(3):38–43, 1999.

[Pov00] D. Povinelli, editor. *Folk Physics for Apes*. Oxford University Press, New York, NY, 2000.

[Rai99] Roope Raisamo. An alternative way of drawing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182. ACM Press, 1999.

[RBP96] A. E. Russon, K. A. Bard, and S. T. Parker, editors. *Reaching into Thought: The minds of the great apes*. Cambridge University Press, Cambridge, UK, 1996.

[RR96] Roope Raisamo and Kiri-Jouko Räihä. A new direct manipulation technique for aligning objects in drawing programs. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 157–164. ACM Press, 1996.

[SH02] Robert St. Amant and Thomas E. Horton. Characterizing tool use in an interactive drawing environment. In *Proceedings of the 2nd international symposium on Smart graphics*, pages 86–93. ACM Press, 2002.

[SH04] Robert St. Amant and Thomas E. Horton. Tool-based direct manipulation environments, 2004.

[Shn83] B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69, 1983.

[Smi87] Randall B. Smith. Experiences with the alternate reality kit: an example of the tension between literalism and magic. In *Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*, pages 61–67. ACM Press, 1987.

[Ste03] K. Sterelny. *Thought in a Hostile World: The Evolution of Human Cognition*. Blackwell Publishers, Oxford, 2003.

[TC97] M. Tomasello and J. Call. *Primate Cognition*. Oxford University Press, New York, NY, 1997.

[TH01] G. Taxén and J. Hourcade. *KidPad 1.0 User's Guide*, 2001.

[Vau96] J. Vauclair. *Animal Cognition*. Harvard University Press, Cambridge, MA, 1996.

[VL96] E. Visalberghi and L. Limongelli. Acting and understanding: Tool use revisited through the minds of capuchin monkeys. In A. E. Russon, K. A. Bard, and S. T. Parker, editors, *Reaching into Though: The minds of the great apes*, chapter 3, pages 57–79. Cambridge University Press, Cambridge, UK, 1996.

[vSv94] L. van Leeuwen, A. Smitsman, and C. van Leeuwen. Affordances, perceptual complexity, and the development of tool use. *Journal of Experimental Psychology: Human Perception and Performance*, 20(1):174–191, 1994.

[WC01] J. B. Wagman and C. Carello. Affordances and inertial constraints on tool use. *Ecological Psychology*, 13:173–195, 2001.

[WC03] J. B. Wagman and C. Carello. Haptically creating affordances: The user-tool interface. *Journal of Experimental Psychology: Applied*, 9:175–186, 2003.