Instrumental Interaction: A Step Beyond Direct Manipulation

Michel Beaudouin-Lafon and Wendy E. Mackay

Abstract Instrumental Interaction is a "generative theory of interaction" that moves beyond the concept of Direct Manipulation in graphical user interfaces to create simpler, yet more powerful forms of interactive systems. The concept of an *instrument* is inspired by observation and theories of how people use physical tools to manipulate objects in the real world. After summarizing the history and principles of Direct Manipulation, we introduce the concept of an *interactive digital instrument* and describe its four generative principles: *Reification*, *Polymorphism*, *Reuse*, and *Currying*. We provide examples that show how designers can create effective instrumental interfaces and discuss directions for future work.

Keywords: Direct manipulation, GUI, Instrument, Reification, Polymorphism, Reuse, Currying, Affordance, Technical reasoning, Co-adaptation.

Michel Beaudouin-Lafon
Université Paris-Saclay, 91405 Orsay, France e-mail: michel.beaudouin-lafon@universite-paris-saclay.fr

Wendy E. Mackay
Inria and Université Paris-Saclay, 91405 Orsay, France, e-mail: wendy.mackay@inria.fr

1 Introduction

Billions of people use Graphical User Interfaces, or GUIs, every day, on their smartphones, tablets, laptops, and desktop computers. Although GUIs can be traced back to Ivan Sutherland's (1963) seminal SketchPad program in the early 1960's, it took more than a decade of research at Xerox PARC to develop the Alto and its successors, which culminated in the first commercial graphical workstation, the Xerox Star (Johnson et al., 1989). GUIs did not become widely popular until the 1980s, with the arrival of the first PCs (Personal Computers).

Except for the lack of color and lower resolution, a screenshot from the original Xerox Star (Fig. 1) shows the same "WIMP" interface, based on Windows, Icons, Menus, and Pointing, seen on modern computers. Although the advent of tablets and smartphones forced minor design changes to accommodate touch input and smaller displays, the fundamental principles of GUIs remain unchanged.

Ben Shneiderman first articulated the principles of graphical user interfaces. He coined the term *Direct Manipulation* to contrast with the then-dominant "command-line interfaces" where users interact with objects indirectly by typing syntactically correct command strings and referring to objects such as files by name. Direct Manipulation offers an alternative: Users can point directly to the graphical representation of the object of interest, such as an icon that represents a file, instead of remembering and typing its name. Instead of using their memory to recall object names and command syntax, users need only recognize the relevant objects of interest as they interact with them directly. Some interactions inspired by metaphors from the physical world are indeed very direct, e.g. when moving a window or dragging a file icon to the trash icon. However, many other GUI interactions are actually indirect,

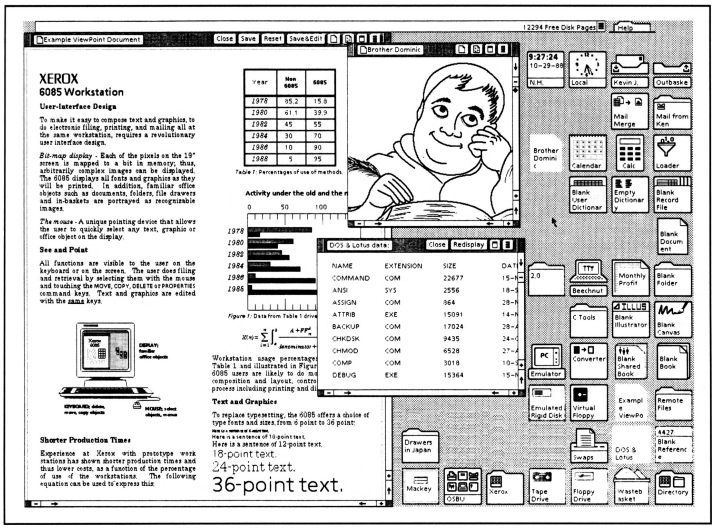

Fig. 1 A screenshot of the original Xerox Star user interface (Needs permission)

because they require the user to access objects of interest indirectly, via menus, dialog boxes, or other secondary interface elements.

Instrumental Interaction (Beaudouin-Lafon, 2000) was inspired by the observation that our interactions in the physical world are often mediated by *tools* rather than performed with our bare hands, which lets us directly manipulate objects of interest. We use pens to write, forks to eat, hammers to drive nails, etc. Human tool use, and tool making, is an incredibly well-developed skill that has evolved over millions of years (Harmand et al., 2015). The goal of Instrumental Interaction is, therefore, to capitalize on these skills and transfer them to the digital world. More generally, Instrumental Interaction seeks to provide both designers and researchers with concepts and principles that enable them to create effective interactive systems that are both powerful yet simple to use (Mackay and Beaudouin-Lafon, 2005).

The rest of this chapter examines the principles of Direct Manipulation and what we mean when we say an interaction is "direct". We then introduce the concept of *interactive digital instrument* and describe the set of supporting theories relating to human tool use that ground and motivate it. Next, we describe the generative principles that enable designers to create instrumental interfaces and provide multiple examples from commercial products and the research literature that illustrate the benefits of an Instrumental Interaction approach. We conclude with a discussion and directions for future work.

2 Direct Manipulation

Ben Shneiderman (1983, p.64)[1] introduced the following four principles of Direct Manipulation:

- Continuous representation of the object of interest.
- Physical actions (movement and selection by mouse, joystick, touch screen, etc.) or labeled button presses instead of complex syntax.
- Rapid, incremental, reversible operations whose impact on the object of interest is immediately visible.
- Layered or spiral approach to learning that permits usage with minimal knowledge.

His article highlights the benefits of Direct Manipulation for novices, who "*can learn basic functionality quickly*"; for experts, who "*can work extremely rapidly to carry out a wide range of tasks*"; and for intermittent users, who "*can retain operational principles*". He also argues that Direct Manipulation reduces the need for error messages since users can see their progress and change direction if needed. They "*experience less anxiety because the system is comprehensible and because actions are so easily reversible*" and they "*gain confidence and mastery because they initiate an action, feel in control, and can predict system responses*" (Shneiderman, 1983, p.64–65).

Shneiderman also identifies several potential problems with Direct Manipulation, all linked to the use of graphical representations. They may not always be appropriate; they may require learning to understand them; they may be misleading; and choosing the right representation may not be easy. Other researchers have identified other issues. For example, Sherugar and Budiu list disadvantages related to each principle on the the popular Nielsen-Norman group website[2]: only a few objects can be represented simultaneously, physical actions may induce repetitive strain injury (RSI), lack of continuous feedback may lead to frustration, and rapid learning depends greatly on the interface design. They also note that Direct Manipulation may be slow, does not support repetitive tasks well, can be more error-prone than typing, and can limit accessibility for users with visual and/or motor impairments.

To better understand the limits of Direct Manipulation, we next address its principles, benefits and critiques, including the definition of "object of interest", the physical actions performed by the user and their effects, and how different categories of users gain proficiency.

---

[1] Later books in the series (Shneiderman, 1997; Shneiderman et al., 2017) changed the first principle to: "*Continuous representation of the objects and actions of interest with meaningful visual metaphors*" and omitted the fourth principle. This chapter retains the original definition from Shneiderman (1983).

[2] https://www.nngroup.com/articles/direct-manipulation/

2.1 What is the object of interest?

The term "object of interest" appears twice in the four principles of Direct Manipulation and several times in the original article (Shneiderman, 1983), yet it is never defined in this nor in any other key publications, such as Hutchins et al. (1985) or Shneiderman (1997). For the purpose of this chapter, we use the following definition:

> The object of interest is the conceptual object that the user manipulates in order to achieve a goal. Conceptual objects are visually represented in the interactive system and are relevant to the current activity.

For example, when writing a report, the object of interest is the digital document being edited; when looking for a file, the object of interest is the hierarchy of files and directories; when searching for a product on an e-commerce website, the object of interest is the product catalog. The object of interest changes over time according to the user's activity: it may shift to a different object, e.g. to a different document; from an object to one of its parts, e.g. from the catalog to a given product; or conversely from an object to its container, e.g. from a file directory to its parent. GUIs provide visual representations of these objects of interest so that users can easily designate them, typically through pointing.

GUIs also feature many visual objects that do *not* represent objects of interest. Users may use *interface objects* — menus, buttons, toolbars, scrollbars, dialog boxes, property sheets (or inspectors) — to directly manipulate objects of interest, but they are not, in themselves, objects of interest in the above sense.

Note that an interactive application may feature objects that *acquire* the status of an object of interest. For example, word processors typically feature *styles* that group together sets of attributes, such as font, color and margins. Applying a style to a paragraph is not only easier than changing each attribute individually, but also ensures consistency across paragraphs within the document. Styles make the interface both simpler — the user may apply a style without knowing the details of its attributes — and more powerful — changing a style attribute affects all paragraphs using that style. Understanding styles is thus a key characteristic of being a proficient word processor user. We refer to styles as *secondary* objects of interest: they complement the primary object of interest, the document in this case, but are also new conceptual objects that users integrate into theirmental model of word processing.

2.2 Direct, or indirect manipulation?

The principles of Direct Manipulation state that objects of interest should be manipulated through physical actions. In cursor-based interfaces, the user points directly at a representation of the object, followed by a click or a drag[3]. A click has an immediate effect on the object, whereas a drag either combines two objects, e.g., dragging an

[3] Curiously, Shneiderman's second principle mentions pressing buttons, which seems inconsistent with the spirit of Direct Manipulation.

object to the trash icon, or controls a continuous change in an attribute of the object, e.g., increasing the object's size.

However, in practice, very few Direct Manipulation interfaces involve acting directly on the object of interest. For example, consider how to change a text block's font size with a standard graphical editor. A typical interface (Fig. 2a) requires the user to first "select" the object of interest by clicking on it. Next, the user searches for the desired command, usually represented as a menu item located in the "tool bar", and selects it. This command opens a dialog box. The user must now find the appropriate text box, type the desired font size, and finally click on the OK button, at which point the text changes to the specified size. Only the first action in this interaction sequence is "direct" — all others involve interaction objects that are not the object of interest. This violates both the second principle of Direct Manipulation, since users must "master a complex syntax" and the third principle, since the operations are neither incremental nor reversable and "whose impact on the object of interest" is *not* immediately visible.

An alternative interface (Fig. 2b) might feature an inspector[4]. Here, the user first selects the object, which in turn displays an inspector showing the object's properties. The font size is represented by a slider, which, when moved, immediately adjusts the target object's size. Although closer to satisfying the second and third principles, this interaction is still not direct. One could argue that the inspector acts as an alternative representation of the object of interest, but this is inconsistent with the first principle of Direct Manipulation since the representation is not continuous: it shifts to another object as soon as the user selects a different target.

The most direct form of interaction for this example (Fig. 2c) would be to click on the object to display its corner handles, and then drag a handle to increase the font size. However, even in this case, one could argue that the user does not directly manipulate the object itself, but rather the object's handles. Moreover, it is difficult to imagine a similar direct interaction that would let the user change the object's color instead of its size.

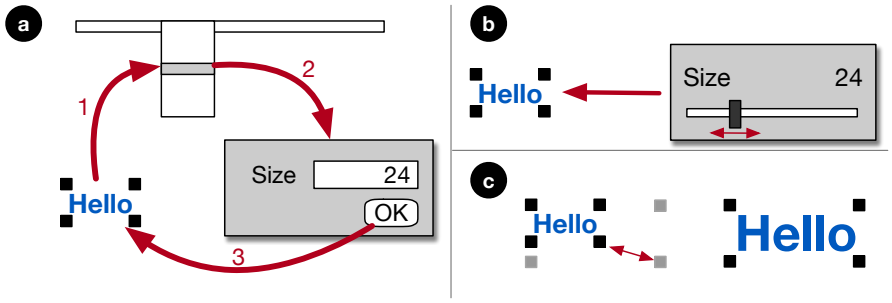

Fig. 2 Three versions of "direct" manipulation: (a) menu and dialog box; (b) inspector or property sheet; (c) resize handles

[4] Also called "property box", inspired by the Xerox Star's "property sheet".

The goal of this example is to show that the "direct manipulation" of current GUIs is actually very *indirect*: users must shift attention from their object of interest to external interface objects, which violates the principles of Direct Manipulation.

2.3 Object-verb vs. verb-object syntax

Each of the above interactions begins with the user selecting an object of interest by pointing at it directly. This *object-verb* syntax is a very common in GUIs: first identify the object of interest, then specify which command to apply, typically by selecting it from a menu, and finally specify the command's parameters, if any. This approach is particularly effective if the user wants to apply multiple commands to the same object, since the object typically stays selected after a command has been applied to it.

However, many GUIs also feature the opposite *verb-object* syntax, usually in the form of a tool palette that contains a set of icons. Clicking an icon activates the corresponding tool, which the user can then apply to one or more objects of interest. Here, the command is specified first, then the object. The command parameters are either part of the tool itself, in which case they can typically be changed via an inspector, and/or are specified when applying the tool, such as when defining the size of an object being created with a drag interaction. This approach is particularly effective when the user wants to apply the same command to multiple objects, in sequence. It is also useful when creating a new object of interest, since one cannot use the object-verb syntax if the object does not yet exist.

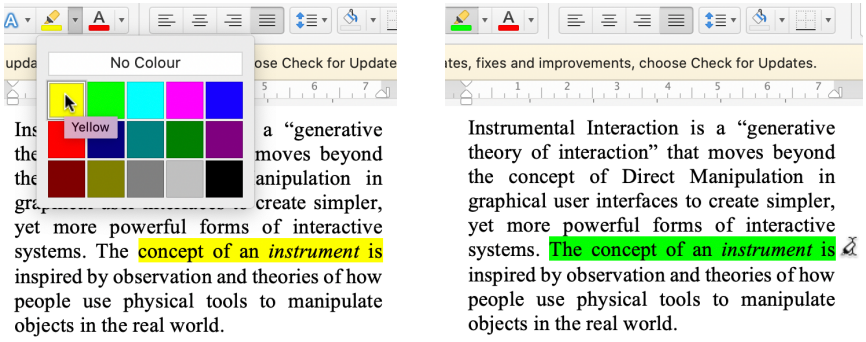

Fig. 3 The highlighter tool in Microsoft Word uses the object-verb syntax (left) when text is selected and the verb-object syntax when no text is selected (right).

In some (rare) cases, both syntaxes coexist for the same command and even through the same interface object. For example, in Microsoft Word, the highlighter toolbar button follows the object-verb syntax when there is currently selected text i.e. it highlights that text when clicked (Fig. 3a), and the verb-object syntax when only

the insertion cursor appears, with no text selected. Here, the highlighter becomes a brush that "paints" a highlight as the user clicks and drags over the text (Fig. 3b).

Featuring both verb-object and object-verb syntax in the same user interface may confuse users. Even so, as mentioned above and demonstrated by Mackay (2002), each syntax can have advantages, depending on the task at hand. The object-verb syntax is more efficient for object-centric tasks, whereas the verb-object syntax is more efficient for command-centric ones. Unfortunately Direct Manipulation does not provide a solution for addressing this conflict.

2.4 Power vs. simplicity

One goal of Direct Manipulation is to support *simplicity of interaction*, with a correspondingly layered approach to learning the interface. One way to understand this principle is that "*simple things should be simple, complex things should be possible.*"[5] Many simple things are indeed simple in GUIs, due to the "point-and-click" nature of basic interactions and the standardization of various common commands such as open/close, scroll, resize, and copy-paste. Yet many common commands that should be simple are difficult to discover or hard to use, including managing styles in word processors, aligning objects in graphics tools and specifying search criteria on e-commerce websites. For example, some CAD systems feature hundreds of commands and some business applications include dozens of screens with hundreds of forms. Direct Manipulation interfaces have difficulty scaling to this level of complexity.

A second goal should be to support *power of expression* to help users perform complex tasks. Early critiques of Direct Manipulation highlighted the lack of support for repetitive tasks and the user's inability to create their own commands. Unlike language-based command-line interfaces such as the Unix shell, where users can use pipes and create scripts that combine existing commands, Direct Manipulation interfaces lack the ability to abstract over objects and commands. Several Direct Manipulation interfaces address this limitation by letting users create scripts, e.g. Automator[6] or Shortcuts[7] on MacOS. However, Direct Manipulation applications need simpler mechanisms for personalizing or creating new commands.

2.5 Summary

Although Direct Manipulation is the dominant interaction style on personal computers, it has a number of its limitations — not so much with its principles, but rather

[5] Quote attributed to Alan Kay.

[6] https://support.apple.com/en-gb/guide/automator

[7] https://support.apple.com/en-gb/guide/shortcuts-mac

in how they have been interpreted and implemented over the years. In order to move beyond these limitations, we summarize our critique by asking how can we:

- extend the concept of the "object of interest"?
- expand Direct Manipulation to embrace indirect interaction?
- define a richer syntax of interaction?
- scale Direct Manipulation to larger numbers of objects and commands?
- increase the power of Direct Manipulation interfaces?

3 Instrumental Interaction

The analysis of Direct Manipulation in the previous section shows that, while the principles spelled out by Shneiderman (1983) are sound, they are not followed by the vast majority of today's GUIs. One reason is that the large number of possible commands available for any non-trivial object of interest simply cannot be made available through direct interaction with that object, i.e. by just pointing at it. Rather than trying to achieve directness, we should instead embrace the fact that interaction is often indirect and make this indirection explicit.

Instrumental Interaction (Beaudouin-Lafon, 2000) is an interaction model based on the observation that our interactions in the physical world are often mediated by tools and are rarely direct, i.e. with just our bare hands. For example, while we might use our fingers to write in the sand, we more often write text with tools such as pens or keyboards. Instrumental Interaction transfers this concept of mediated interaction into the digital world and gives the mediators, called *instruments*, the status of first-class objects in the interface. Unlike menu commands, which use verbs to describe their actions, instruments transform the interface's capabilities for action into objects that can be manipulated by the user through direct action. Before describing instruments in greater detail, we review supporting theories of human tool use to help understand the power of tools.

3.1 Theoretical foundations

Humans are unique in their ability to actively create and use tools. Human tool use can be traced back at least 3.3 million years (Harmand et al., 2015), and involves a set of cognitive processes studied by psychologists. The theories of affordances and technical reasoning are especially valuable for helping us to understand how tools help humans expand their capabilities for action.

James Gibson (1979) defines affordances as properties of an object that enable specific actions by animals and humans. He observes that: "*When in use, a tool is a sort of extension of the hand, almost an attachment to it or a part of the user's own body, and thus is no longer a part of the environment of the user.*" (Gibson, 1979, p.40). Understanding the capabilities of the environment requires *perceiving* these affordances, which we learn through a process that Eleanor Gibson (1969) calls *perceptual learning*. Subsequent research has shown that holding a tool makes it an extension of our body schema (Klatzky et al., 2008). Holding a tool therefore redefines the affordances of the environment for the person holding it, letting her perceive new capabilities for action. For example, holding a pen reveals the surfaces upon which the pen may work.

Conversely, looking for a way to draw on a surface reveals the mark-making capabilities not only of a pencil, but also of a charred piece of wood. In other words, we are experts at figuring out which object or combination of objects could be used as a tool for a particular purpose. On the one hand, we use our knowledge of

the purposes of a tool to use it in appropriate circumstances, e.g. fetching a pencil for writing. This *procedural knowledge* is acquired through repetition and does not require understanding the underlying mechanisms at play. On the other hand, we use *technical reasoning* (Osiurak, 2014), a specific type of reasoning where we use our knowledge of abstract technical laws and the perceived properties of an object to use it as a tool. For example, we know that a sharp object can pierce a softer one, and we can apply this knowledge to use a pencil to poke a hole in a sheet of paper.

Recent work has shown evidence that technical reasoning is indeed at play when interacting with digital content (Renom et al., 2022, 2023), involving what the authors call *interaction knowledge*, i.e. knowledge of the "laws" of digital content and GUIs. We also have extensive procedural knowledge of how to use GUIs, acquired through repeated use. This provides a strong basis for using tool-mediated interaction in GUIs. It also suggests going beyond the simple tools used in some interfaces, such as the brush in a painting program, to support more powerful tools that can be appropriated by users (Mackay, 2000), to provide tools that can manipulate other tools, and to let users create their own tools.

3.2 What is an instrument?

The key characteristic of an instrument is to *mediate* interaction with the object of interest: the user interacts with the instrument, and the instrument interacts with the object of interest (Figure 4). More precisely, the instrument receives *input actions* from the user, such as moving the mouse, clicking, or entering text. These actions may produce *reactions* from the instrument, such as changing the cursor or providing haptic feedback. The instrument transforms the user actions into *commands* sent to the target object. These commands generate a *response*, such as a visible change of state of one or more objects of interest, the creation or deletion of an object of interest, or a side effect, e.g., printing a document. The response may also be interpreted by the instrument and conveyed back to the user. For example, the instrument may continuously update the display showing the object's current size as the user moves the cursor.

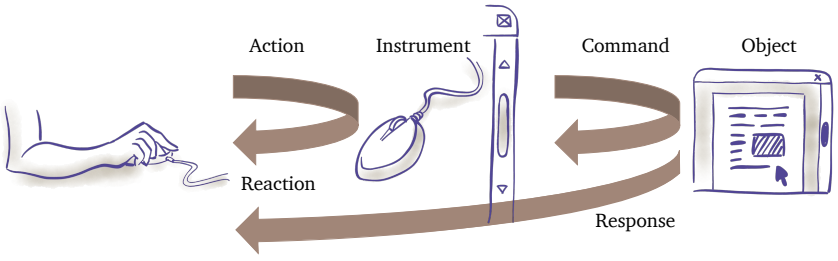

Fig. 4 Mediated interaction through an instrument.

Consider a scrollbar — a navigation instrument designed to access the parts of a document that are not currently visible. When the user clicks on one of the scrollbar's arrows or drags its "thumb", the scrollbar sends the document a scrolling command, and the document responds by updating its view. The scrollbar reacts to the user's actions by highlighting either the arrow or the thumb. The scrollbar also responds to the scrolling commands by updating the thumb's position to reflect the new view.

Now consider an instrument for creating oval shapes in a graphics editor. The user first clicks the instrument to "grab" it. When the user then presses the mouse button, the instrument sends a creation command to the canvas which creates a small oval. As the user drags the mouse, the instrument sends resize commands to the oval; in response, the oval changes size and the instrument displays the current size. Alternatively, the oval may be created only when the user releases the mouse button. In this case, it is the instrument that displays the oval shape in reaction to the user moving the mouse to adjust the size; the creation command is sent only at the end.

A third, less common example, is a color-changing tool with multiple controls, such as a collection of color swatches, RGB or HSV color sliders, and/or a color wheel. The user selects a target object by clicking on it and then operates the relevant controls to modify its color. The rest of this section describes and illustrates two taxonomies of instruments and a set of instrument properties to better characterize the design space of Instrumental Interaction.

3.2.1 Modes of operation

The examples above illustrate the three primary modes of operation for an instrument:

- *In-hand*: the instrument must first be "grabbed" by the user who can then apply it to one or more objects. For example, the oval-creation tool described above or a drawing tool's paint brush must be grabbed before use. These instruments typically live in tool palettes and are represented by icons. Selecting an icon grabs the instrument, ideally changing the mouse cursor to the instrument's icon in order to convey the fact that future actions will be interpreted by the instrument.
- *Attached* (also called *Embedded*): the instrument is bound to an object of interest obviating the need to specify that object when using the instrument. To make the connection explicit, the instrument is displayed within or next to the object of interest. For example, both the afore-mentioned scrollbar and the window's "close" button are bound to that window: a single click or drag applies the instrument to the window. Handles for transforming geometric shapes are also embedded instruments.
- *Remote*: the instrument operates on an object that must be specified separately and that can be visually distant from the instrument. For example, the color tool described above and, more generally, current GUI property inspectors that operate on selected object(s) are all remote instruments.

The main difference among these three modes of operation is how the object of interest is specified. This raises the possibility of changing an instrument's mode

of operation. For example, turning the attached close-window instrument into an in-hand instrument creates a closing tool that can first be grabbed, then applied to several windows by clicking anywhere inside them. Conversely, attaching a color instrument to a target object would bind it to that object, e.g. as a small icon displayed next to the object, similar to the handles that transform graphical shapes. Clicking that icon reveals the control panel and lets the user change the color. Different color panels could then be displayed simultaneously, making it easy to copy-paste colors between color panels. Alternatively, pressing the mouse button on the icon and dragging the mouse could control two components of the color property, such as the saturation and value, and the scroll wheel could control the hue. Another example could deattach the scrollbar from one document, turning it into a remote instrument that affects multiple documents, enabling synchronized scrolling across them.

3.2.2 Types of instruments

We present the following taxonomy of instruments derived from typical GUI commands organized according to their effects on the user's objects of interest. Although not meant to be exhaustive, this taxonomy covers the majority of commands used in today's GUIs. Instruments can:

- create new objects.
- delete existing objects.
- change properties of objects.
- annotate objects.
- organize and combine objects.
- manage constraints among objects.
- navigate among objects.

.

Creating a new object typically requires specifying its initial properties and relationships to other objects, either when the tool is applied, e.g., when dragging a bounding box to specify the size of a graphical shape, or when copying it from a default or current set of properties, which may then be modified through an inspector. A more instrumental approach would embed these properties into the tool itself: The instrument would set the object's properties after it is created (see below) or set the creation tool's properties for later use. Combined with the ability to duplicate existing tools, users can now create their own palettes of cloneable template objects.

Deleting objects may take advantage of the move instrument to drag objects to a trash icon. An alternative "vacuum cleaner" instrument might directly delete the objects it hovers over. Both approaches feature a temporary holding object — the trash or the vacuum cleaner — that preserves the deleted objects for later inspection and retrieval, if necessary.

Changing object properties is probably the most frequent activity in most GUIs, especially in content-creation applications. Handles are attached instruments that users can drag to control geometric properties such as position, size or rotation.

Handles can also affect non-geometric properties, e.g. using a drag action to control one or more parameters, such as opacity or contrast/brightness. Alternatively, an instrument can be attached to an inspector window that includes controls for modifying the object's properties. Instead of providing a single inspector window associated with the currently selected object, a more instrumental approach would let the user manage multiple inspectors, each attached to an object or a group of objects. The object could include a handle (attached instrument) that opens the inspector or an in-hand instrument that opens a new inspector when applied to an object of interest. In-hand instruments would let users select multiple objects, e.g. by circling them. In this case, the inspector would display multiple values for each property, as in ManySpector (Hoarau and Conversy, 2012). Supporting multiple simultaneous inspectors lets users copy and paste values from one inspector to another, e.g. with a simple drag-and-drop action.

Annotating objects lets users communicate information to other users and/or document decisions or design rationale. Users should be able to attach comments, tags and other metadata to any object. Tags and other metadata can help users refind objects when used together with the navigation instruments described below. Annotation instruments should let users define the annotation's scope — document, object of interest, set of objects or part of an object, e.g. a text selection — and then the annotation's content. Annotations should be treated as first-class objects that can be manipulated in the same way as other objects. This lets them become secondary objects of interest when, for example, the user wants to annotate an annotation. Users should also be able to modify the scope of an annotation, rather than deleting and recreating it.

Organizing and combining objects occurs in applications that deal with large numbers of objects or very complex objects. In these cases, objects are often organized into a hierarchy that can be viewed as a list or a tree. Instruments for managing such lists are often embedded in the list, such as the triangles for opening and closing a level in the hierarchy or handles for reordering the list's content. In-hand instruments can also be useful for creating groups and adding or removing objects from them, especially when objects appear as a collection on a canvas, such as icons in a file manager or shapes in a graphics editor. Users typically create groups by selecting a set of objects and then invoking the "group" command. However, this becomes cumbersome and error-prone when the user has to ungroup, re-select and re-group just to add or remove an object. A grouping instrument would let users directly add or remove objects from a group, e.g. through a drag operation.

Managing constraints among objects involves setting relationships that are maintained as the corresponding objects change. For example, a spreadsheet formula establishes a relationship between a set of source cells and cell's value, which is updated when the source cells involved in the formula change. We can consider the spreadsheet's formula editor as an instrument for editing formulas, where users can select source cells by name or through direct selection. The formula editor highlights the relevant source cells, which the user can then modify simply by dragging one end of the highlighted range. More generally, creating and editing a constraint involves dealing with multiple objects, which can lead to potentially complex interactions.

Representing constraints as first-class, interactive objects, or creating instruments designed for manipulating constraints, produces simpler yet more powerful interactions. For example, the StickyLines alignment instrument (section 5.5) makes it easy to align and distribute graphical objects. Not only is the alignment and distribution maintained when individual objects are added, removed or dragged, but users can also tweak the alignment and distribution parameters to produce a desired result.

Navigation helps users access objects that are not currently visible, often via attached instruments such as scrollbars, pan-and-zoom controls and links. Users can also type keywords into a search instrument. By extension, navigation also includes filtering the objects to be viewed, e.g. when querying an e-commerce website. Filtering is often achieved by asking the user to fill out a form and then sending the query, with few possibilities to refine the query after the fact, short of re-submitting it. An instrumental approach replaces the form with a set of instruments or an inspector that specify the value or range of interest, letting the user adjust the filters interactively, as with Dynamic Queries (Shneiderman et al., 1992). Filtering can also use the tags managed with the annotation tools described above.

Note that since instruments are objects, the above instruments can also be applied to other instruments. Users should be able to create, delete, organize and configure instruments with the same or similar instruments that they use to manipulate primary objects of interest. Treating instruments as secondary objects of interest will in turn allow users to perform technical reasoning with them.

3.2.3 Properties of instruments

Beaudouin-Lafon (2000) introduces the following set of properties to characterize instruments, based on their mode of operation and the mapping between the user's actions and their effect on the object of interest:

- Degree of indirection;
- Degree of integration;
- Degree of compatibility;

These properties can be used to compare instruments and to explore the design space by looking for instruments that achieve the same effect but have a different degree of indirection, integration and/or compatibility.

The *degree of indirection* measures the spatial and temporal offsets of an instrument. The *spatial offset* is the distance on the screen between the instrument and the object it controls. In-hand instruments have a zero spatial offset as they act directly on the object of interest. Embedded instruments such as handles have a small spatial offset, while remote instruments have an arbitrarily large one. Note that a large spatial offset is not necessarily undesirable. Just as locating a light switch next to the door is more convenient than placing it directly on a ceiling light, it often makes sense to place instruments in a well-known, easily accessible location on screen, especially if they act on multiple objects simultaneously.

The *temporal offset* is the time difference between physical actions to the instrument and the object's response. Following the third principle of Direct Manipulation, a short temporal offset helps create a sense of causality. This is why indirect manipulation through, e.g., dialog boxes, should be avoided. If a large temporal offset is necessary, e.g. for performance reasons, a proxy representation of the effect of the command should show the instrument's reaction. For example, if an object cannot be resized at interactive rates, a rubber-band shape should provide feedback to the user. Another example involves drag-and-drop operations that only perform the action at drop time, resulting in potentially large temporal offsets. Here, the instrument's reaction should provide *feedforward*, that is, show what will happen if the user drops the object.

The *degree of integration* measures the proportion of the input device's number of degrees of freedom (DoF) that are actually used by the instrument. We note this as: $di = x \rightarrow y$ where x is the DoF of the input device and y the DoF used by the instruments. A mouse provides two degrees of freedom, which are both used when, e.g., resizing an object ($di = 2 \rightarrow 2$). A scrollbar, on the other hand, only uses one dimension and therefore has a lower degree of integration ($di = 2 \rightarrow 1$).

This leaves room for more powerful instruments such as Orthozoom (Appert and Fekete, 2006), which controls both scrolling and zooming ($di = 2 \rightarrow 2$). Bi-manual input, multi-touch input, and pen-based input (which captures pressure and pen orientation) provide more degrees of freedom and therefore enable richer instruments. For example, the pinch-to-zoom gesture (4 DoF) on a touch surface controls translation (2 DoF), rotation (3 DoF for center and angle) and scaling (1 DoF) with the movements of two fingers ($di = 4 \rightarrow 6$). With a single mouse, this requires either multiple instruments, or using modifier keys to switch mode within an instrument. Note however that a single mouse (2 DoF) can be used to control both translation and rotation (Beaudouin-Lafon, 2001; Kruger et al., 2005) ($di = 2 \rightarrow 5$). These examples show that some techniques let users control more degrees of freedom than those of the input device, albeit with some limitations such as longer or less intuitive interactions.

The *degree of compatibility* measures the similarity between the user's actions and the object's response. Resizing an object by dragging a corner handle has a high degree of compatibility since the object follows the movements of the cursor. By contrast, scrolling with a scrollbar has a low degree of compatibility because moving the thumb downwards moves the document upwards. Many music applications operate rotary knobs with a left-right or up-down dragging motion, which has a low degree of compatibility. Some word processors ask users to enter a number in a text field to specify a margin, which has a lower degree of compatibility than dragging a tab in a ruler. Note however that performing linear movements with the mouse is easier and more precise than circular movements (Nancel et al., 2011) and that entering a number may be more accurate if the user must specify an exact value.

Note that none of these properties suggest optimal values — higher or lower is not necessarily better. Instead, the best alternative is subject to a depends on set of trade-offs. Sometimes, it may be better to offer users multiple instruments that

accomplish the same task. They can then pick the one they prefer, either because it is more suitable or efficient for the task at hand, or simply because it is more familiar.

3.3 Summary

Instruments are objects that can be manipulated by the user to affect other objects, leading to a mediated form of interaction that builds on our skills in the physical world. The examples in this section are all drawn from current GUIs, showing that in many cases the indirect interactions identified in the previous section can be interpreted as mediated by instruments. However, Instrumental Interaction lets designers reinterpret these interactions, leading to simpler yet more powerful designs.

These particular examples show the power of separating instruments from the objects they operate on. Pushing this idea further, one could imagine that tools are not trapped inside applications as they are today. For example, many current applications feature one or more commands for assigning colors to objects. Users must constantly adapt to each and cannot easily copy colors across applications. With an instrumental approach, there is no reason why the color instrument described earlier could not be used in *any* application, configured to control multiple target objects from different applications, e.g. to ensure that a text document's color matches that of a diagram.

Furthermore, users should be able to exert technical reasoning and use instruments in unanticipated ways. For example, many objects feature colors that cannot be changed by the user, such as the background color of a document or window. Ideally, a color instrument should be able to change these colors as well. In other words, the ability of an instrument to manipulate an object should be determined by the *properties* of the object and the corresponding capabilities of the instrument. In the same way that a pencil can write on different types of surfaces, even those it was not necessarily designed for, such as a wall, a digital pen should work with different types of content.

While designing new instruments can be inspired by similar parallels with the physical world, Beaudouin-Lafon and Mackay (2000) have developed a principled approach that was later extended into the concept of generative theory of interaction (Beaudouin-Lafon et al., 2021). The next section describes the generative principles of Instrumental Interaction.

4 Generative Principles for Instrumental Interaction

Beaudouin-Lafon et al. (2021) describe generative theories of interaction as theories that "*draw insights from empirical theories about human behavior in order to define specific concepts and actionable principles, which in turn serve as guidelines for analyzing, critiquing and constructing new technological artifacts.*" The article describes Instrumental Interaction as such a theory, based on the theories of human tool use outlined in the previous section. The new concept is the *instrument*, and the three principles are *Reification*, *Polymorphism* and *Reuse*, which were first introduced by Beaudouin-Lafon and Mackay (2000)[8]. We also add a new principle, *Currying*, which is briefly mentioned by Beaudouin-Lafon (2004).

4.1 Reification

Reification is the process by which an abstract concept is turned into a concrete object. In Instrumental Interaction, the concepts are the commands and the concrete objects are instruments. In other words, an instrument reifies an abstract command, i.e. it embodies a command into an object that the user can interact with to manipulate another object, called the target object. For example, a scrollbar reifies the action of navigating a document into an object that the user interacts with to act on the document.

Reification is the fundamental generative principle for creating new instruments: when confronted with the problem of providing a given functionality, designers can imagine an instrument that reifies it, in order to make it concrete for the user. This contrasts with the traditional approach where a new functionality is typically bound to a new menu item that acts as a "magic word" to invoke the functionality, resulting in users having to learn which command does what and when it is available.

A properly designed instrument leverages technical reasoning: the instrument should have a technical effect that makes it more understandable and memorable than the mere arbitrary mapping between a menu entry and a command. However, *Reification* alone does not guarantee success. Applying the principle requires skill, creativity, and an iterative process to refine the design and test it with users.

4.1.1 Polymorphism

In computing, polymorphism is the capability of a function to work with parameters of different types. In Instrumental Interaction, *Polymorphism* is the principle that helps create instruments that can be applied to objects of different types.

[8] This section borrows and adapts the descriptions of these principles from both Beaudouin-Lafon and Mackay (2000) and Beaudouin-Lafon et al. (2021).

Most current interfaces feature polymorphic commands such as open, cut, copy, paste, save or delete, which can work with, e.g., documents, text, images, or sound. These commands could easily be turned into polymorphic instruments. Instrumental Interaction encourages *Polymorphism* in order to enable technical reasoning: the more object types an instrument can work with, the more likely it can be used in powerful and/or unexpected ways. For example, a coloring instrument should work with different types of objects, and even different parts of an object such as its border, background or text, thereby replacing the various color commands of current systems. The same coloring instrument could also be applied to another instrument, such as a brush or paint bucket, thereby defining the color it applies to other objects. Check where else we used this example to avoid redundancies.

Polymorphism enables powerful interactions with groups of objects. An instrument designed to be applied to a single object, such as the color instrument, can be applied to a group of objects. In this case, it is applied to each object that understands the instrument. If the instrument is polymorphic, it will apply to a wider set of object types, and therefore can be used with heterogeneous groups.

Polymorphism complements *Reification*: *Reification* results in the creation of new instruments whereas *Polymorphism* helps reduce the number of instruments by "factoring" them. Reducing the number of instruments is important if we want to keep the interface simple while increasing its power.

As in programming languages, *Polymorphism* can be used for better or for worse. To be successful, a polymorphic instrument must have some internal consistency that makes its behavior with different types of objects predictable. This leverages technical reasoning rather than require procedural learning of different commands. For example, it is easier to infer that the color instrument can change the background color of a document rather than learn and remember that it can only be changed by going to the document settings. Check with the other place where we mention this.

4.1.2 Reuse

Reuse is the ability for users to reuse previous actions (*Input Reuse*) or the results of these actions (*Output Reuse*). *Input Reuse* supports repetitive tasks, while *Output Reuse* supports the common practice of starting from existing content and modifying it rather than starting from scratch.

Output Reuse is already present in many interfaces through, e.g., the copy-paste and duplicate commands. *Reification* augments the power of *Output Reuse* by creating new objects (the instruments) that can be reused. For example, the ability to duplicate and then modify instruments lets users create multiple variants, such as brushes with different patterns.

Similarly, *Input Reuse* is often present in the form of a redo command, and sometimes through the ability to access and selectively re-execute past actions listed in a history panel. *Polymorphism* augments the power of *Input Reuse* by letting users apply the same instrument to objects of different types.

4.2 Currying

Building on the parallel with functional programming, using an instrument can be considered as the application of a function that takes as parameters one or more objects of interest and other values, such as the color to be applied by a color tool, or the position and size of a shape being created. In functional programming, currying, also called partial application, transforms a function f of n parameters $f(x_1, x_2, ...x_n)$ and a value v into a function f' of $n-1$ parameters by fixing the first parameter of f to v: $f'(x_2, ...x_n) = f(v, x_2, ...x_n)$. A generalizing of currying consists of defining a new function where any number of parameters of the original function are fixed to given values.

Currying an instrument consists of creating a new instrument where one or more parameters of the original instrument are fixed. For example, an instrument that sets a color takes as parameters the target object that will receive the color and the value of the color. Currying the target object parameter creates an attached instrument that is bound to that object. Currying the color instead creates an instrument that assigns that color to any object it is applied to. The value of currying lies in letting users decide what parameters they want to fix, thereby customizing their environment. For example, a user can create a set of color instruments for each color of the palette they want to work with, as when an artist picks a set of color pens.

4.3 Power in combination

As shown in the above examples, each of these principles are useful for both analyzing existing interfaces and creating new designs. However, it is their combination that makes them even more powerful.

- *Reification* and *Polymorphism*: GUIs typically features hundreds of commands, therefore reifying commands can lead to a large number of instruments. Polymorphism helps reduce this complexity by merging multiple instruments with similar functionality into a single one. For example, the same color instrument can be used to change the border, fill and text color of an object simply by applying it to different parts of the object.
- *Polymorphism* and *Input Reuse*: Polymorphism expands the types of objects that can be targeted by a given instrument. This facilitates input reuse, since the same action can be applied to objects of different types, as well as to groups of such objects. For example, a resize instrument can be applied to a group of different shapes.
- *Reification* and *Output Reuse*: Reification creates new objects, not just new instruments. These new objects, which typically become secondary objects of interest, can then be reused through, e.g., copy-paste. For example, the concept of a collection of attributes can be turned into an object called a style, represented as such in the interface. A style can then be copied to create a new style and modify it.

- *Output Reuse* and *Currying*: When applied to instruments, output reuse enables duplicating existing instruments, which can then be specialized through currying. For example, the user may create several copies of a brush instrument and specialize each of them with a different brush shape and paint color.
- *Currying* and *Reification*: Reification creates new instruments, which can then be curried to create more specific ones. Currying encourages the definition of instruments requiring a larger number of parameters, since these more powerful instruments can then be specialized. For example, rather than creating different instruments for turning text bold, italic and underlined, a single instrument controlling all three properties can be curried to create three instruments that control each property, but also other instruments that control other combinations, such as bold italic or bold underlined.

5 Examples

This section describes a number of examples from the literature and from our own work that illustrate the principles of Instrumental Interaction, even though some of them were published before these principles were articulated.

5.1 Tool-based interactions

Many content-creation applications features tool palettes with icons representing conventional tools such as brush, pen, or eraser. These tools match our description of in-hand instruments: they must be grabbed by selecting them in the tool palette before using them. While intuitive, this does not match our experience of using tools on a physical desk: we grab a tool, then drop it nearby to grab another one so that we can quickly grab the first one again. The fact that tools are stuck in tool palettes in digital applications forces many round trips to the palette.

LocalTools (Bederson et al., 1996) solves this problem by letting users drop the tool they are holding with a double-click. The tool then stays on the work surface where it was dropped and can be picked up again with a single click. LocalTools were designed for use by children and were found to be easier to use than a tool palette.

Raisamo and Räihä (1996) describe an alignment stick that is used to align objects by pushing them, much as one would use a ruler in the physical world. A modifier key is used to move the stick "above" the objects so as to not push them. A user can thus push a set of objects from below, then move the stick above them and push them again from the top. Raisamo (1999) expand on this idea by providing other tools for sculpting objects with carving, shrinking and cutting sticks.

These examples show that a tool-based approach can lead to simpler and more expressive interactions. Children and novice users are more likely to understand a tool-based interface than one based on commands hidden in menus. At the same time, in-hand tools encourage the development of expertise: like physical brushes and carving tools, digital brushes and carving sticks require precise gestures, whereas commands usually rely on parameter setting with sliders or text fields.

5.2 Bimanual tools: Toolglasses

When we use in-hand tools in the physical world we often use our two hands to, e.g., hold the object in one hand and apply the tool with the other. Guiard (1987) theorized the asymmetric role of the hands in bimanual action, whereby the non-dominant hand acts first and sets the context for the dominant hand to act. This arrangement is at play when we use combinations of tools, such as positioning a

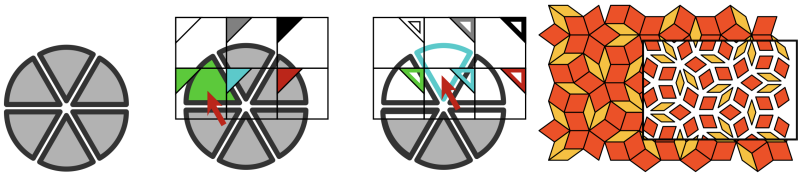

Fig. 5 Left: Toolglass for setting fill and border color; Right: MagicLens that shrinks objects to make them more visible (Need permission - https://www.billbuxton.com/tgml93.html and https://dl.acm.org/doi/10.1145/166117.166126)

ruler with the non-dominant hand to draw a straight line with a pencil held in the dominant hand.

Toolglasses (Bier et al., 1993) apply this principle to GUIs by turning tool palettes, called toolglasses, into movable palettes controled by the non-dominant hand. The non-dominant hand moves the toolglass over the object of interest and the dominant hand then activates a tool in the palette with a "click-through", i.e. a click that determines both the tool to apply and the object to apply it to. The prototype demonstrates a wide range of tools for actions such as coloring, setting text attributes, or copy-pasting. MagicLenses complement toolglasses by transforming the representation of objects, such as reversing the z-order, turning an image into black-and-white or displaying a wireframe. Combining toolglasses and magic lenses enhances their power, for example by letting users act on hidden parts of objects. Studies show that toolglasses can be up to 40% faster than classical static palettes.

Toolglasses demonstrate the power of using tools in combination and taking advantage of our natural skills in coordinating bimanual action. The CPN2000 (Beaudouin-Lafon and Lassen, 2000; Beaudouin-Lafon and Mackay, 2000) Colored Petri Nets editor combined toolglasses with static palettes, so that users could choose what worked best for them. Any static palette could be turned into a semi-transparent toolglass by clicking it with the non-dominant hand. CPN2000 also implemented a bi-manual static palette, whereby the user grabbed a tool by clicking in the palette with the non-dominant hand, and applied the tool with the dominant hand. This proved to be even more efficient than toolglasses for some tasks, as it did not require users to coordinate the placement of the toolglass over the object of interest to apply a tool to it. The important lesson of this work however was to provide users with a choice of interaction techniques so they could use the ones they were most familiar, comfortable or efficient with.

5.3 Currying commands: SideViews

SideViews (Terry and Mynatt, 2002) provide "*on-demand, persistent, and dynamic previews of commands*": as the cursor hovers over a menu command or toolbar icon, a live preview of the effect of that command on the current document is displayed next to the cursor. While this technique is not instrumental, it features an interesting

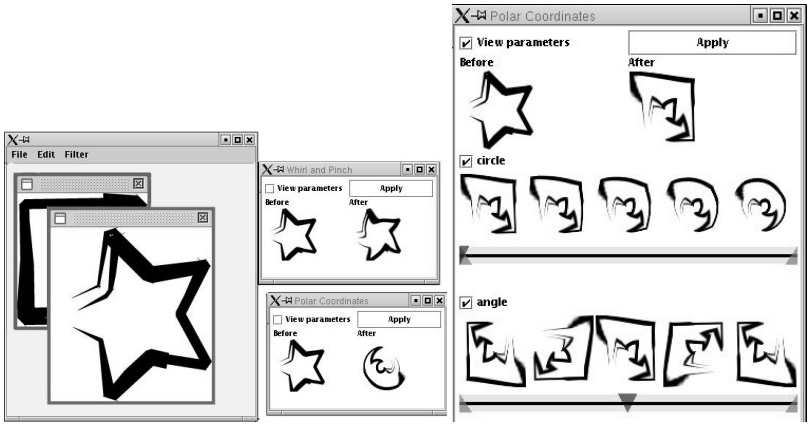

Fig. 6 Left and center: SIDEVIEWS showing the preview of two different filters; Right: preview of the effect of two parameters of a filter (Need permission - https://dl.acm.org/doi/10.1145/571985.571996)

combination of reification and currying that could be used with instruments. The reification consists of turning the preview into a persistent window by clicking its title bar. The preview window can then be moved around and closed when not needed any more.

Currying occurs at two levels. First, the persistent preview windows are bound to the command used to create them. This means that the user can create other persistent previews for different commands. At the same time, the persistent previews are bound to the current document: opening another document updates all the persistent previews with that document; the user can therefore quickly switch among different documents to see the effect of the same commands on different content.

Second, when a command uses additional parameters, such as different controls for an image filter, the preview displays a range of results by sampling the parameter space. The user can then control the range of each parameter to narrow down the sampled space. The slider controlling each parameter is therefore an instrument that controls this parameter while the other parameters and the target document are fixed.

5.4 Property instruments: OBJECT-ORITENTED DRAWING

OBJECT-ORIENTED DRAWING (Xia et al., 2016) is a tablet application for vector-based drawing that reifies the properties of the objects into small interactive tiles attached to the object (Fig. 7). Each tile is in effect an attached instrument that can be used to change the value of the corresponding property by tapping it. A pinch-out gesture expands the tile into a sequence showing the history of values of that property, making it easy to perform local undo. Additional interactions let users link together the values of different tiles, e.g. so that two objects share the same color. It is also

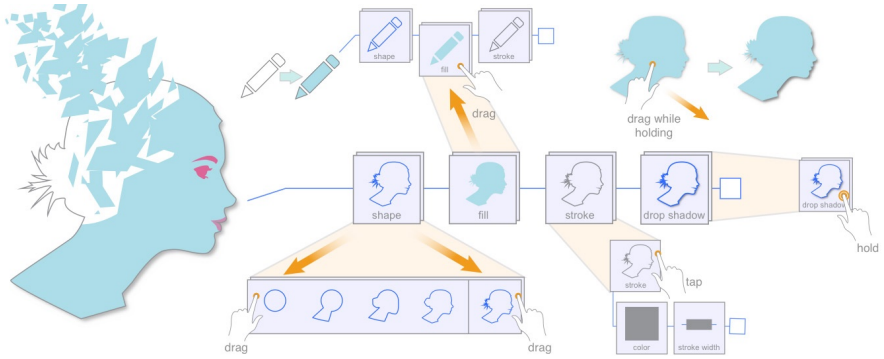

Fig. 7 The different interaction techniques for manipulating attribute objects in Object-Oriented Drawing (Need permission - https://dl.acm.org/doi/10.1145/2858036.2858075)

possible to interpolate between two values by tapping two tiles of the same type simulateneously: the two tiles are linked and a third tile appears between them that can be moved to show the interpolated value (Fig. 8).

Object-Oriented Drawing demonstrates the power of reification and of treating the reified objects as instruments, i.e. as objects with agency over other objects. Each tile not only shows the value of a property but also provides a rich set of action for changing it, accessing its history, linking it to other objects, or combine it with other values through interpolation. Attaching tiles to objects rather than using a single property inspector provides additional power, e.g. for copying or linking properties, or simply for visually comparing them. Finally, multitouch and bimanual interaction on a touchscreen expand the vocabulary of actions and improve the power and expressivity of the interface. For example, linking properties is achived by tapping the source tile and, while still touching it, tapping the target of the link; Creating an interpolator instrument is achieved by simulataneously tapping two tiles of the same type.

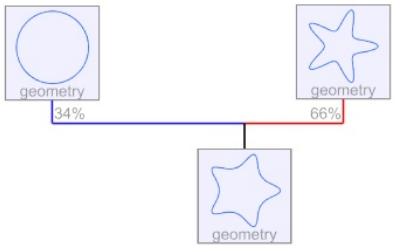

Fig. 8 Interpolation between two shapes in Object-Oriented Drawing. The tile at the bottom can be moved left to right to control the mix between the two shapes. The same technique works with other properties, e.g. color (Need permission - https://dl.acm.org/doi/10.1145/2858036.2858075)

5.5 Alignment instrument: StickyLines

Most graphical tools feature a set of commands for aligning and distributing graphical objects horizontally or vertically along their centers or one of their sides. StickyLines (Ciolfi Felice et al., 2016) reifies these commands into a single instrument that creates a new type of object: a magnetic guideline (Fig. 9) [9]. Graphical objects can be attached to the guideline by moving them close to it, and removed from the guideline by moving them away from it. The objects move with the guideline when it is dragged, as if they were glued to it. The guideline therefore acts as an instrument to manipulate the objects attached to it. The guideline is also an object of interest that represents the group of objects: applying an instrument to the guideline, such as a color tool, applies it to all the objects on the guideline.

The guideline has a number of parameters to control its behavior:

- Distribution may be enabled, in which case the objects move along the guideline to maintain regular spacing;
- When distribution is enabled, the user can change the function mapping the rank of the object to its position. By default the function is linear but can be changed to, e.g., a sigmoid so that the objects on the left and right are further apart than those in the middle (Fig. 11);
- Each object can be "tweaked", i.e. offset from its default snapping position. This is useful to adjust the visual alignment of objects, e.g. when layout out logos with very different shapes (Fig. 10a). Similarly, the bounding box of each object can be tweaked, which affects the layout when distribution is enabled (Fig. 10b).

The performance evaluation of StickyLines showed up to 40% performance improvement over traditional align and distribute commands (Ciolfi Felice et al., 2016). This is due to the fact that StickyLines keep objects aligned, and that users can manipulate aligned objects as groups even when individual objects are attached to multiple guidelines. StickyLines are demonstrably both simpler to use and more powerful than traditional align/distribute commands.

A recent version of Adobe Illustrator features a similar concept, called the "Objects on path" tool [10]. The user selects a set of object, a path, and clicks the "Objects on path" tool, which snaps the objets to the path, As with StickyLines, the objects stay attached to the path as it is manipulated, and they are re-distributed along the path when an object is added to / removed from the path. The path also features controls (i.e., embedded instruments) for rotating all the attached objects simultaneously and for changing the extent of the path used for the layout.

[9] Magnetic guidelines were first introduced in the CPN2000 project mentioned earlier (Beaudouin-Lafon and Lassen, 2000; Beaudouin-Lafon and Mackay, 2000).

[10] https://helpx.adobe.com/illustrator/using/objects-on-path.html

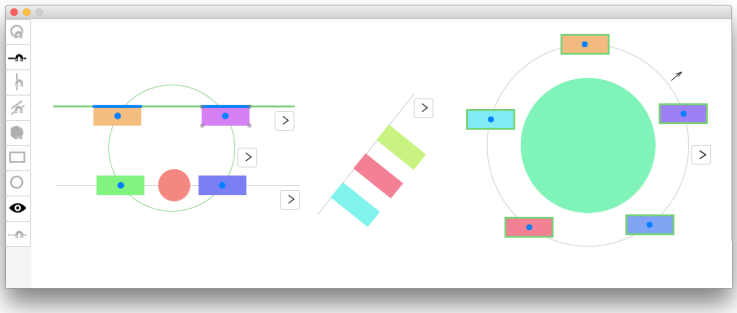

Fig. 9 Circular and linear StickyLines (left), parallel StickyLine (center) and Stickyline created from an existing shape (right) and resized. The user attaches shapes to a StickyLine by moving them onto it (By permission - image from the authors)

5.6 Text instruments: TEXTLETS

Word processors are one of the most heavily used applications, yet text editing has not changed very much over the past decades. Most commands in word processors are based on text selection: the user selects some text, invokes a command that affects the text, e.g. to turn it bold, and the selection disappears as soon as the user clicks elsewhere. So if the user wants to operate on text that had been selected before, they have to select it again.

TEXTLETS (Han et al., 2020) is based on the idea of turning transient text selections into persistent objects: the user selects text and creates a textlet, which appears in a sidebar; Clicking the textlet re-selects the corresponding text. The power of TEXTLETS comes from the fact that they are also remote instruments that operate on their associated text. For example, a *countlet* is a textlet that counts words (Fig. 12 and displays the word count in real time, as the text is edited, while a *variantlet* is a textlet that lets the user define several alternatives for the text and pick wich one to display. A more sophisticated behavior is the *searchlet* (Fig. 13), which dynamically

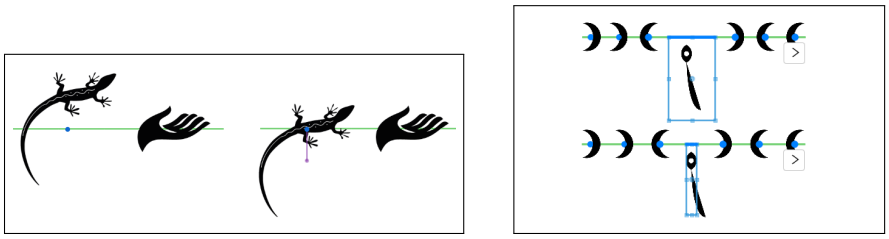

Fig. 10 Left: Tweaking the position (purple line) for visually correct alignment; Right: Tweaking the bounding box (blue box) for visually correct distribution (By permission - image from the authors)

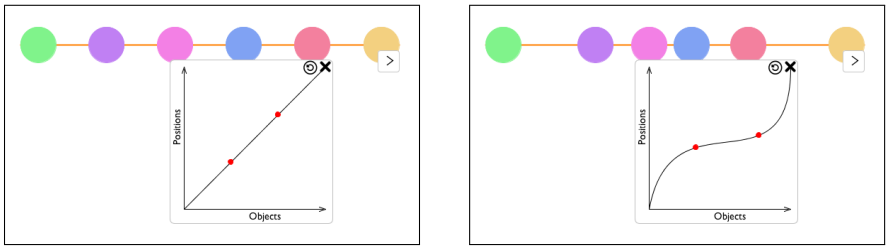

Fig. 11 Controlling the distribution of objects along a StickyLine. The red dots control the shape of the curve and therefore the spacing between adjacent shapes on the guideline. Left: default (linear) mapping; Right: sigmoid mapping to make objects in the middle closer together (By permission - image from the authors)

creates textlets for each occurrence of the search text and lets users replace them one by one, in the order they want, or all at once. Replaced occurrences are displayed in a different color, and the user can undo and redo any change.

A study found TEXTLETS particularly easy to use and efficient, giving users a sense of control and appropriate feedback on their actions (Han et al., 2020). Users also found unexpected uses, such as using a searchlet to automatically highlight words that they had decided not to use in their text.

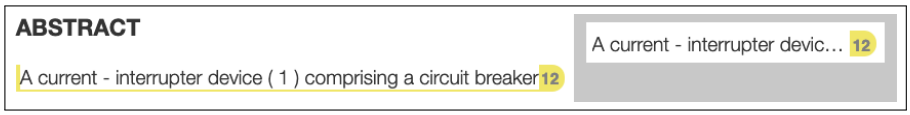

Fig. 12 Textlet for counting words. The word counter appears both in the text itself and in the side panel (By permission - image from the authors)

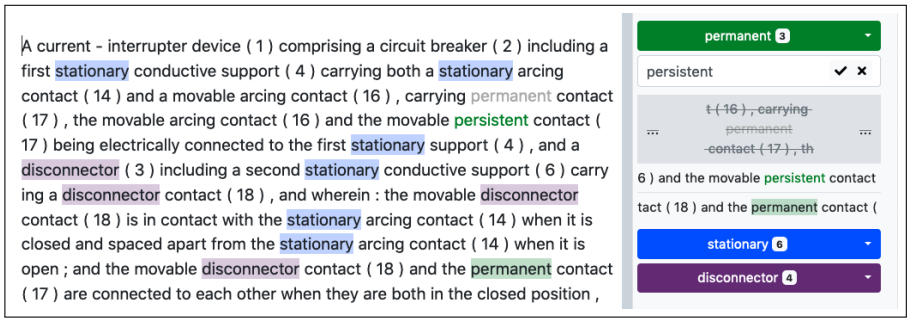

Fig. 13 Three textlets for searching and replacing text. The side panel lets users specify the replace string, exclude matches from a global replace, and perform and undo global and individual replacements. In the document, matching occurrences are highlighted with the same color as the textlet, e.g. "stationary"; replaced occurrences use the color of the textlet, e.g. "persistent"; excluded matches are in grey, e.g. "permanent" (By permission - image from the authors)

6 Discussion

We concluded section 2 with five questions to be addressed in order to go beyond the limitations of Direct Manipulation. How can we:

- extend the concept of the "object of interest"?
- expand Direct Manipulation to embrace indirect interaction?
- define a richer syntax of interaction?
- scale Direct Manipulation to larger numbers of objects and commands?
- increase the power of Direct Manipulation interfaces?

We now address these five questions, and then discuss the limitations of Instrumental Interaction and outline future work.

6.1 Beyond Direct Manipulation

6.1.1 Extending the concept of object of interest

Instrumental Interaction extends the notion of object of interest by creating new objects called instruments. While these instruments are meant to be "transparent" to the user (see below), they are also objects and can become the user's focus, as when a pencil needs sharpening and becomes the user's object of interest, and the pencil sharpener the tool. This is further enabled by the use of *Currying*, which turns parameters of the commands issued by an instrument into properties of that instrument. These properties can thus be changed by other instruments, such as when changing the color of a brush. Instruments therefore can and should become secondary objects of interest.

Moreover, the use of *Reification* leads to the creation of new objects that are not instruments. We cited the example of styles, which users of word processors must understand in order to use the system efficiently. Other examples include the layers used in many drawing and image editing applications, or the processing nodes used in the node-link diagrams of some procedural tools for describing the sources, filters and other processing steps applied to generate output, such as video effects in the Da Vinci Resolve video editor[11] or 3D content in the Blender modelling system[12].

Creating more such objects of interest lead to creating new instruments (or making existing instruments more polymorphic), which in turn may lead to identifying new objects of interest. While this can lead to unwanted complexity (see below), turning abstracts concepts into objects and instruments makes the interface more concrete and facilitates learning.

[11] https://www.blackmagicdesign.com/products/davinciresolve

[12] https://docs.blender.org/manual/en/latest/interface/controls/nodes/introduction.html

6.1.2 From indirect to mediated interaction

Instrumental Interaction is by definition indirect: rather than acting directly on the object of interest, the user acts on an instrument, which acts on the object. Yet, because it is based on a familiar form of manipulation in the physical world, the tool held in the hand becomes "transparent", i.e. it becomes an extension of the user's body, at least until the tool breaks down or does not work as expected. In this case, the focus turns to the instrument, which becomes the object of interest (see above).

Instruments create modes: grabbing an instrument enters a new interaction mode where users' actions have a different effect than in other modes. Modes are often considered harmful, but even Larry Tesler, a strong opponent to modes, recognized that "*modes can be good when they support a metaphor like picking up a brush*" (Tesler, 2012). Indeed, instruments reduce the risks of mode errors because they are *ready-at-hand*, i.e. they are internalized by the user.

While this internalization is arguably at play when using in-hand instruments, what about embedded and remote instruments? With embedded instruments, the sense of engagement results from the closeness between the instrument and the target object and the immediate effect of manipulating the instrument, as if it were "pulling strings" on the target object. With remote instruments, the sense of engagement is less strong because the connection between the instrument and the target object is less explicit. Unless the interface provides visual feedback, the user must remember this connection, increasing cognitive load.

Like in-hand instruments, embedded and remote instruments create modes. However, whereas in-hand instruments create *temporal* modes, i.e. modes that are enabled while the instrument is in-hand, embedded and remote instruments create *spatial* modes, i.e. modes that are enabled while the cursor is over the instrument and interacts with it. Depending on the context of use, one type of mode, and therefore one type of instrument, may be more appropriate or effective for the user. Designers should therefore provide different instruments for the same task, or let users transform an in-hand instrument into an embedded or remote one, or vice-versa. The key property of mediated interaction, however, is that actions on the instrument should have a visible, immediate effect on the target object. In this sense, mediated interaction, and therefore Instrumental Interaction, satisfies the third principle of Direct Manipulation.

6.1.3 Enriching the syntax of interaction

The previous discussion has highlighted that different types of instruments work differently. In-hand instruments must be first picked up, and then the target object must be specified, while embedded and remote instruments can be used directly, since their target object is pre-determined. These two different syntaxes do not seem to be a problem for users, probably because they correspond to natural actions in the physical world: taking a tool in-hand, vs. acting on a control embedded into the

object, such as setting the temperature on an oven, vs. acting at a distance, such as using a light switch to turn on a ceiling lamp.

The syntax of interaction can be enriched by using both hands. Many GUIs use keyboard modifiers to affect the action of the tool controlled by the cursor. Toolglasses (Bier et al., 1993) use the non-dominant hand to position the instrument and the dominant one to act. Object-Oriented Drawing Xia et al. (2016) uses multitouch input from one or both hands to enable powerful commands such as linking and interpolating.

More generally, our natural hand dexterity is poorly used by GUIs. Increasing the number of degrees of freedom captured by the input device and putting them to good use, as discussed in section 3.2.3, obviates the need for complex syntax where the result of a command depends, e.g., on the order in which objects have been selected. Instrumental Interaction can help "deconstruct" complex commands into simpler ones. For example, aligning objects in a traditional interface requires selecting the objects to align, then the alignment type. If one object was omitted, adding it to the alignement requires reselecting the objects and re-aligning them. Often times, this moves all the objects as there is no way to know which reference point is used by the align command. By contrast, StickyLines uses one action to create a new guideline, and then the standard drag action to move objects onto/away from the guideline. Having more objects to interact with makes it easier to leverage simple actions for complex effects.

6.1.4 Scaling up to large numbers of objects and instruments

Turning commands into instruments and other concepts into objects can lead to an increased complexity of the interface. Menus and toolbars of current GUIs are already crowded, and typical interfaces feature a number of panes, panels and floating windows. As mentioned earlier, *Polymorphism* can help reduce the number of instruments by making them polymorphic. However this is unlikely to completely solve the problem, especially since *Currying*, on the other hand, leads to the creation of more specific instruments.

Looking at the kitchen of a chef, the workbench of a woodworker or the workshop of a painter shows evidence that any sophisticated activity requires a large number of objects and tools. The difference between these activities and the use of a desktop application is that in the physical world, the expert taylors their environment for their needs by organizing the space, selecting the subset of tools needed for a particular task, and reconfiguring the space when switching activity. By contrast, the layout of desktop applications and the content of their toolbars and menus is usually fixed and cannot be changed, or only marginally. Some systems feature several "personas" corresponding to different activities, such as de-rushing, video editing, sound editing and color-grading in a video production system. Selecting a persona reconfigures the interface for the corresponding activity, but the personas and the corresponding layout are usually predefined.

Instrumental Interaction encourages users to pick the instruments and objects they work with. Not only the instruments but also the toolbars they live in, and therefore the entire layout of the interface, should be configurable. Users should be able to easily import new instruments, create new ones through *Currying* or by combining existing instruments together, and more generally taylor the entire interface to their needs. These configurations should themselves be first-class objects that can be saved and recalled, maybe in the form of personas. Externalizing these configurations would allow a user to work on someone else's computer and instantly adapt the interface to his or her practice.

6.1.5 Increasing power and simplicity

We have illustrated a number of situations where the use of Instrumental Interaction leads to interfaces that are simpler to use yet more powerful than traditional GUIs. For example, StickyLines makes it easier to align objects than traditional alignment commands, but also makes it more powerful because the alignement can be maintained when adding/removing objects, and aspects of the alignement such as the distribution of objects or the "tweaks" to their position or size, can be controlled by the user.

Increased simplicity stems from the fact that physical actions are easier to understand and learn than the vocabulary of commands typically found in menus. Some actions, such as snapping an object to a guideline that can then be moved, qualify as "one-try-learning", i.e. seeing it once is enough to understand it and being able to use it. Simplicity also relies on our ability to perceive affordances of digital objects and our knowledge of the underlying "technical laws" for manipulating them (see section **??**).

Increased power stems from the ability to embed more complex behaviors into instruments than in traditional commands. For example, the tiles of Object-Oriented Drawing embody not only the current value of a property but also its previous values, providing instant access to the history and the ability to revert to a previous state. Increased power can also result from the ability to create new instruments by combining existing ones. For example, MagicLenses can be combined with Toolglasses simply by superposing them, enabling the user to interact with the hidden parts of an object revealed by the magic lens.

The generative principles help combine power and simplicity in various ways. *Reification* increases both power and simplicity by creating new objects for the user to manipulate. *Polymorphism* increases simplicity by reducing the number of instruments, and increases power by resulting in instruments that can be applied to a wider range of objects. *Currying* increases simplicity by letting users create specialized tools. *Input Reuse* increases power by capturing previous input, which can be used to create new instruments, similar to macros and scripts in command-line interfaces. *Output Reuse* increases simplicity by making it possible to reuse content rather than having to re-create it.

6.2 Limitations and future work

Instrumental Interaction is based on a metaphor of tool-use and is therefore well-suited to interfaces where objects can be represented visually and their properties manipulated through appropriate tools. Instrumental Interaction is *not* adapted to dialogue-based interfaces, where the user issues commands in a natural or artificial language, as in command-line interfaces or the recent wave of prompt-based interfaces for interacting with generative AI systems such as ChatGPT[13] or MidJourney[14]. Compared to command-line interfaces, Instrumental Interaction lacks the ability to abstract over objects and commands and therefore of a language to create new instruments, similar to how Linux users can create their own scripts. This is clearly an area for research on instruments for creating instruments, e.g. using programming-by-demonstration (Cypher et al., 1993) or other approaches. Regarding prompt-based interaction, we note that recent work such as DirectGPT (Masson et al., 2024) has focused on combining prompts with more direct manipulations. We expect this trend to continue and provide users with more instrumental interfaces to AI systems.

Another challenge for Instrumental Interaction is movement-based interfaces, which are often based on free-hand and/or full-body gestures. Gesture-based interfaces are common on touch-based devices such as tablets and smartphones, where the gestures are performed directly on the surface. Free-hand and full-body gestures are common in Virtual Reality (VR) and Augmented Reality (AR) interfaces. While some gestures, such as pinch-to-zoom on a tablet or pinch-to-grab in an AR/VR interface, can be seen as instrumental, others are symbols that the user needs to learn and learn to perform, such as three-finger pinch or four-finger swipe. The fact that a touch-based interface lets users litterally touch the content makes it more difficult to design a mediated interaction, although Object-Oriented Drawing shows that it is indeed possible. In AR/VR environments, the use of tools seems natural and is common, but the lack of haptic and kinesthetic feedback can make physical interactions with objects and tools deceptive. Further work is therefore needed to investigate appropriate design principles for Instrumental Interaction in gesture-based environments.

Tangible interfaces, on the other hand, are a natural area for expanding Instrumental Interaction beyond desktop interfaces. Graspable interfaces (Fitzmaurice et al., 1995) demonstrated the value of using physical objects for manipulating digital ones, and the Reactable[15] is a good example of an actual product based on physical manipulation of blocks representing sound-processing instruments that can be connected together. Not sure whether to cite this and what else to say: TouchTokens (Morales González et al., 2016) bridge the gap between tangible and touch-based interaction...

Finally, Instrumental Interaction needs software toolkits and frameworks to facilitate the implementation of instrumental interfaces on today's platforms. Current user

[13] https://chatgpt.com

[14] https://www.midjourney.com

[15] https://reactable.com

interface toolkits are centered on the notion of "widget", a self-contained interactive object such as a button, text field or menu. Instruments break many of the assumptions built into existing toolkits because they are not self-contained: by definition, they interact with another object which may be anywhere on the screen and in the in-memory model of the interface. Ideally, instruments should even work *across* applications rather than being trapped inside each running application. This breaks the heavy barriers set up by operating systems and the applications themselves, e.g. the difficulty for two tabs of the same web browser to communicate together. Fully embracing Instrumental Interaction would require a deep redesign of the software stack going from the operating system all the way to the graphics and user interface layers of today's computers. Projects such as WEBSTRATES (Klokmose et al., 2015) provide a glimpse into a world without applications, where content and tools can be freely mixed and appropriated by users. In the meantime, however, instrumental toolkits cite Stratify?? should be developed to bring Instrumental Interaction to standard desktop and tablet interfaces.

7 Summary and Conclusion

Future work: AI + Instrumental Interaction

Acknowledgements This work was partially supported by European Research Council (ERC) grants #321135 "CREATIV: Creating Co-Adaptive Human-Computer Partnerships" and #695464 "ONE: Unified Principles of Interaction".

References

Appert C, Fekete JD (2006) Orthozoom scroller: 1d multi-scale navigation. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '06, p 21–30, DOI 10.1145/1124772.1124776, URL https://doi.org/10.1145/1124772.1124776

Beaudouin-Lafon M (2000) Instrumental interaction: An interaction model for designing post-wimp user interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '00, pp 446–453, DOI 10.1145/332040.332473, URL http://doi.acm.org/10.1145/332040.332473

Beaudouin-Lafon M (2001) Novel interaction techniques for overlapping windows. In: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology, ACM, New York, NY, USA, UIST '01, p 153–154, DOI 10.1145/502348.502371, URL https://doi.org/10.1145/502348.502371

Beaudouin-Lafon M (2004) Designing interaction, not interfaces. In: Proceedings of the Working Conference on Advanced Visual Interfaces, ACM, New York, NY, USA, AVI '04, p 15–22, DOI 10.1145/989863.989865, URL https://doi.org/10.1145/989863.989865

Beaudouin-Lafon M, Lassen HM (2000) The architecture and implementation of cpn2000, a post-wimp graphical application. In: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology, ACM, New York, NY, USA, UIST '00, p 181–190, DOI 10.1145/354401.354761, URL https://doi.org/10.1145/354401.354761

Beaudouin-Lafon M, Mackay WE (2000) Reification, polymorphism and reuse: Three principles for designing visual interfaces. In: Proceedings of the Working Conference on Advanced Visual Interfaces, ACM, New York, NY, USA, AVI '00, pp 102–109, DOI 10.1145/345513.345267, URL http://doi.acm.org/10.1145/345513.345267

Beaudouin-Lafon M, Bødker S, Mackay WE (2021) Generative theories of interaction. ACM Transactions on Computer-Human Interaction 28(6), DOI 10.1145/3468505, URL https://doi.org/10.1145/3468505

Bederson BB, Hollan JD, Druin A, Stewart J, Rogers D, Proft D (1996) Local tools: an alternative to tool palettes. In: Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology, ACM, New York, NY, USA, UIST '96, p 169–170, DOI 10.1145/237091.237116, URL https://doi.org/10.1145/237091.237116

Bier EA, Stone MC, Pier K, Buxton W, DeRose TD (1993) Toolglass and magic lenses: the see-through interface. In: Proceedings of the 20th Annual

Conference on Computer Graphics and Interactive Techniques, ACM, New York, NY, USA, SIGGRAPH '93, p 73–80, DOI 10.1145/166117.166126, URL https://doi.org/10.1145/166117.166126

Ciolfi Felice M, Maudet N, Mackay WE, Beaudouin-Lafon M (2016) Beyond snapping: Persistent, tweakable alignment and distribution with stickylines. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology, ACM, New York, NY, USA, UIST '16, pp 133–144, DOI 10.1145/2984511.2984577, URL http://doi.acm.org/10.1145/2984511.2984577

Cypher A, Halbert DC, Kurlander D, Lieberman H, Maulsby D, Myers BA, Turransky A (eds) (1993) Watch what I do: programming by demonstration. MIT Press, Cambridge, MA, USA

Fitzmaurice GW, Ishii H, Buxton WAS (1995) Bricks: laying the foundations for graspable user interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press/Addison-Wesley Publishing Co., USA, CHI '95, p 442–449, DOI 10.1145/223904.223964, URL https://doi.org/10.1145/223904.223964

Gibson EJ (1969) Principles of Perceptual Learning and Development. Appleton & Co

Gibson JJ (1979) The ecological approach to visual perception. Houghton, Mifflin and Company, Boston, Massachusetts

Guiard Y (1987) Asymmetric division of labor in human skilled bimanual action. Journal of Motor Behavior 19(4):486–517, DOI 10.1080/00222895.1987.10735426

Han HL, Renom MA, Mackay WE, Beaudouin-Lafon M (2020) Textlets: Supporting constraints and consistency in text documents. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '20, p 1–13, DOI 10.1145/3313831.3376804, URL https://doi.org/10.1145/3313831.3376804

Harmand S, Lewis JE, Feibel CS, Lepre CJ, Prat S, Lenoble A, Boës X, Quinn RL, Brenet M, Arroyo A, Taylor N, Clément S, Daver G, Brugal JP, Leakey L, Mortlock RA, Wright JD, Lokorodi S, Kirwa C, Kent DV, Roche H (2015) 3.3-million-year-old stone tools from Lomekwi 3, West Turkana, Kenya. Nature 521(7552):310–315, DOI 10.1038/nature14464, URL https://doi.org/10.1038/nature14464

Hoarau R, Conversy S (2012) Augmenting the scope of interactions with implicit and explicit graphical structures. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '12, p 1937–1946, DOI 10.1145/2207676.2208337, URL https://doi.org/10.1145/2207676.2208337

Hutchins EL, Hollan JD, Norman DA (1985) Direct manipulation interfaces. Human-Computer Interaction 1(4):311–338, DOI 10.1207/s15327051hci0104_2, URL https://doi.org/10.1207/s15327051hci0104_2

Johnson J, Roberts T, Verplank W, Smith D, Irby C, Beard M, Mackey K (1989) The Xerox Star: A retrospective. Computer 22(9):11–26, DOI 10.1109/2.35211

Klatzky RL, MacWhinney B, Behrman M (2008) Embodiment, ego-space, and action. Series: Carnegie Mellon Symposia on Cognition Series, Psychology Press, New York, NY, USA

Klokmose CN, Eagan JR, Baader S, Mackay W, Beaudouin-Lafon M (2015) Webstrates: Shareable dynamic media. In: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, Association for Computing Machinery, New York, NY, USA, UIST '15, p 280–290, DOI 10.1145/2807442.2807446, URL https://doi.org/10.1145/2807442.2807446

Kruger R, Carpendale S, Scott SD, Tang A (2005) Fluid integration of rotation and translation. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '05, p 601–610, DOI 10.1145/1054972.1055055, URL https://doi.org/10.1145/1054972.1055055

Mackay W (2000) Responding to cognitive overload: Co-adaptation between users and technology. Intellectica 30(1):177–193, DOI 10.3406/intel.2000.1597

Mackay WE (2002) Which interaction technique works when? floating palettes, marking menus and toolglasses support different task strategies. In: Proceedings of the Working Conference on Advanced Visual Interfaces, ACM, New York, NY, USA, AVI '02, p 203–208, DOI 10.1145/1556262.1556294, URL https://doi.org/10.1145/1556262.1556294

Mackay WE, Beaudouin-Lafon M (2005) Generative approaches to simplicity in design. In: International Forum: Less is More - Simple Computing in an Age of Complexity, Microsoft Research, Cambridge, UK

Masson D, Malacria S, Casiez G, Vogel D (2024) Directgpt: A direct manipulation interface to interact with large language models. In: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '24, DOI 10.1145/3613904.3642462, URL https://doi.org/10.1145/3613904.3642462

Morales González R, Appert C, Bailly G, Pietriga E (2016) Touchtokens: Guiding touch patterns with passive tokens. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA, CHI '16, p 4189–4202, DOI 10.1145/2858036.2858041, URL https://doi.org/10.1145/2858036.2858041

Nancel M, Wagner J, Pietriga E, Chapuis O, Mackay W (2011) Mid-air pan-and-zoom on wall-sized displays. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA, CHI '11, p 177–186, DOI 10.1145/1978942.1978969

Osiurak F (2014) What neuropsychology tells us about human tool use? the four constraints theory (4ct): Mechanics, space, time, and effort. Neuropsychology Review 24(2):88–115, DOI 10.1007/s11065-014-9260-y

Raisamo R (1999) An alternative way of drawing. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '99, p 175–182, DOI 10.1145/302979.303035, URL https://doi.org/10.1145/302979.303035

Raisamo R, Räihä KJ (1996) 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, ACM, New York, NY, USA, UIST '96, p 157–164, DOI 10.1145/237091.237113, URL https://doi.org/10.1145/237091.237113

Renom MA, Caramiaux B, Beaudouin-Lafon M (2022) Exploring technical reasoning in digital tool use. In: Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '22, DOI 10.1145/3491102.3501877, URL https://doi.org/10.1145/3491102.3501877

Renom MA, Caramiaux B, Beaudouin-Lafon M (2023) Interaction knowledge: Understanding the 'mechanics' of digital tools. In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '23, DOI 10.1145/3544548.3581246, URL https://doi.org/10.1145/3544548.3581246

Shneiderman B (1983) Direct manipulation: A step beyond programming languages. Computer 16(8):57–69, DOI 10.1109/MC.1983.1654471, URL https://doi.org/10.1109/MC.1983.1654471

Shneiderman B (1997) Designing the User Interface: Strategies for Effective Human-Computer Interaction, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Shneiderman B, Williamson C, Ahlberg C (1992) Dynamic queries: database searching by direct manipulation. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '92, p 669–670, DOI 10.1145/142750.143082, URL https://doi.org/10.1145/142750.143082

Shneiderman B, Plaisant C, Cohen M, Jacobs S, Elmqvist N, Diakopoulos N (2017) Designing the User Interface: Strategies for Effective Human-Computer Interaction, 6th edn. Pearson, Hoboken, NJ, USA

Sutherland IE (1963) Sketchpad: a man-machine graphical communication system. In: Proceedings of the May 21-23, 1963, Spring Joint Computer Conference, ACM, New York, NY, USA, AFIPS '63 (Spring), p 329–346, DOI 10.1145/1461551.1461591, URL https://doi.org/10.1145/1461551.1461591

Terry M, Mynatt ED (2002) Side views: persistent, on-demand previews for open-ended tasks. In: Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology, ACM, New York, NY, USA, UIST '02, p 71–80, DOI 10.1145/571985.571996, URL https://doi.org/10.1145/571985.571996

Tesler L (2012) A personal history of modeless text editing and cut/copy-paste. Interactions 19(4):70–75, DOI 10.1145/2212877.2212896, URL https://doi.org/10.1145/2212877.2212896

Xia H, Araujo B, Grossman T, Wigdor D (2016) Object-oriented drawing. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '16, p 4610–4621, DOI 10.1145/2858036.2858075, URL https://doi.org/10.1145/2858036.2858075