Interaction Substrates: Combining Power and Simplicity in Interactive Systems

Wendy E. Mackay
mackay@lisn.fr
Université Paris-Saclay, CNRS, Inria
Laboratoire Interdisciplinaire des Sciences du Numérique
91400 Orsay, France

Michel Beaudouin-Lafon
mbl@lisn.fr
Université Paris-Saclay, CNRS, Inria
Laboratoire Interdisciplinaire des Sciences du Numérique
91400 Orsay, France

Abstract

Today's graphical user interfaces tend to be either simple but limited, or powerful but overly complex. In order to combine power and simplicity, we introduce *Substrates*, which act as "places for interaction" where users can manipulate objects of interest in a principled and predictable way. Substrates structure and contain data, enforce user-defined constraints among objects and manage dependencies with other substrates. Users can "tune" and "tweak" these relationships, "curry" specialized tools or abstract relationships into interactive templates. We first define substrates and provide in-depth descriptions with examples of their key characteristics. After explaining how Substrates extend the concept of Instrumental Interaction, we apply a *Generative Theory of Interaction* approach to analyze and critique existing interfaces and then show how using the concepts of Instruments and Substrates inspired novel design ideas in three graduate-level HCI courses. We conclude with a discussion and directions for future work.

CCS Concepts

• **Human-centered computing** → *Human computer interaction (HCI).*

Keywords

Substrates, Content-authoring systems, Creativity Support, Generative Theory of Interaction, Instrumental Interaction, Reification, Polymorphism, Reuse, Currying, Templating, Tuning, Tweaking

ACM Reference Format:
Wendy E. Mackay and Michel Beaudouin-Lafon. 2025. Interaction Substrates: Combining Power and Simplicity in Interactive Systems. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26-May 1, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3706598.3714006

1 Introduction

Professional content-authoring applications include complex sets of commands that let experts create sophisticated documents, images, videos and music. However, such applications require significant effort to learn, partially because they often include idiosyncratic techniques that are unique to each application and are rarely transferable to other contexts. They also provide limited capabilities for users to adapt them to their needs and work practices. Li [33] argues that the balance of power between designers and users of creativity-support tools is biased towards the former. Our goal is to provide *designers* with concepts and principles that help them create more flexible digital environments so that *users* enjoy more powerful yet simpler interfaces.

Giving more power to end users is not just a question of adding more features to existing applications — they already have more features than most can deal with [41]. Nor is it a question of facilitating the novice-to-expert transition [12], which usually focuses on improving the user's efficiency rather than challenging the application's feature set or fundamental design. We propose instead to reassess the underlying conceptual model of the *structure* and *content* that users interact with.

Our work is inspired by how people interact with objects in the physical world. People naturally develop an understanding of the properties of objects and of the technical principles for affecting these properties, such as the knowledge that a sharp object can cut a softer one. This *technical reasoning* [45, 46] enables humans to solve problems and achieve their goals in inventive yet simple ways, e.g., when using a napkin as a drawing surface to sketch an idea. This understanding of what diSessa calls "naive physics" [15] does not exist, at least to the same extent, in the digital world, where expertise often simply means developing *procedural knowledge* of the system's idiosyncrasies i.e. "recipes" that do not require a deep understanding of how the system works.

We address these limitations by presenting a novel conceptual model that serves as a paradigm for rethinking the design of graphical user interfaces. We introduce the concept of *Interaction Substrates*, which act as "places for interaction" where users can learn and rely upon easily detectable rules that govern the behavior of digital objects within that space. The resulting interactive systems enable users to manipulate objects in predictable ways, thus encouraging them to develop transferable expertise across applications and over time. We present this approach as a strategy for increasing both the power and simplicity of interactive applications.

We first introduce and define the term Substrate. We then review related research on conceptual models in HCI, document-centric environments, the trade-offs between power and simplicity, and previous uses of the word "Substrate" in HCI. We next describe and illustrate the key characteristics of Substrates. After explaining how we extended Instrumental Interaction [4, 8] to incorporate the concept of Substrates, we introduce two additional principles: *Adjustment* and *Specialization*. We then apply the Generative Theory of Interaction framework [7] to Instruments and Substrates by analyzing and critiquing existing commercial applications and

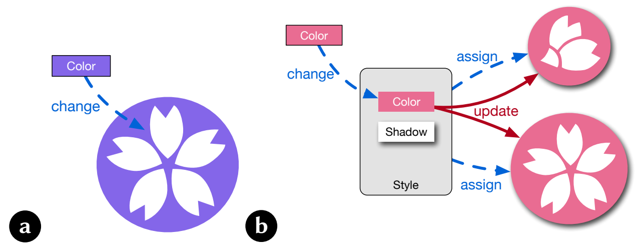


Figure 1: a The user applies commands to change each graphical attribute individually (dashed blue arrow), e.g., turn the background color purple. b Style object: The user creates a style with multiple attributes, e.g., color and shadow, and *assigns* it to two logos (dashed blue arrows). *Changing* the style's color (top dashed blue arrow) automatically *updates* the color of both logos (solid red arrows).

research projects, and describe how using these concepts in three graduate-level courses inspired innovative designs that are both powerful and simple to use. We discuss the benefits and limitations of Substrates and conclude with directions for future research.

2 Definitions

Before proceeding with in-depth descriptions and examples, we first define the key terms used throughout this paper.

2.1 Objects of Interest and Commands

Shneiderman [51] introduced the term *Object of Interest* to describe the conceptual objects that are represented visually in a Direct Manipulation interface. These objects form the *content* that users interact with to achieve their goals. For example, a diagram editor's objects of interest include rectangles, ovals, lines and other graphical shapes, whereas a word processor's objects of interest include sections, paragraphs and figures. Beyond these *primary* objects of interest, interactive applications typically introduce what we call *secondary* objects of interest to make interaction simpler and/or more powerful. For example, "styles" are secondary objects of interest that let experienced users control diagram editors and word processors more efficiently.

Users create, modify and delete Objects of Interest through *commands*. In Graphical User Interfaces (GUIs), users issue commands via *tools* such as menus, buttons, toolbars, scroll bars, dialog boxes and property sheets (or inspectors). Some mimic real-world tools, such as the brush in a painting program. Others resemble control panels, such as the sliders used to select an RBG color. However many just appear as items on a menu, such as the "Print" command. Our goal is to provide a more principled approach to identifying and organizing a coherent set of objects of interest and commands in interactive systems.

2.2 Power and Simplicity

Alan Kay argues that "simple tasks must be simple, and complex ones must be possible" [30]. This sets up a trade-off between power of expression (making complex tasks possible) and simplicity (making simple tasks simple) [38]: a simple interface may lack power, whereas a powerful interface may be too complex to use. We define *simplicity* as the cost of achieving a given result measured as the number of actions such as pointing, clicking or typing [1]. We define *power* as the scope and complexity of a command's effects. For example, a single command that aligns a group of objects is both simpler and more powerful than moving them one by one. Commands that create *relationships* that are then managed by the system are more powerful than commands that simply change the state of an individual object of interest, because they transfer the task of maintaining the relationship from the user to the system. These are also simpler to use, provided that the user understands the relationship and the system provides appropriate feedback.

Style objects illustrate a useful combination of power and simplicity. A style object (Fig. 1) collects the values of multiple attributes and lets the user apply them to any object with a single command. The interaction is thus simpler, since a single action replaces multiple commands, and more powerful, since styles *maintain* the relationships between the shapes and their attached values: changes in any style attribute affects all objects with that style.

2.3 Substrates

We define substrates as places for users to interact with their objects of interest. A substrate:

(1) contains and *structures* objects of interest;
(2) manages internal *constraints* among objects; and
(3) supports *dependencies* from other substrates or external sources.

Designers create substrates to provide a coherent environment where users issue commands that let them interact with both primary and secondary objects of interest, their constraints and dependencies, and the substrates themselves. The following examples illustrate each of the above characteristics:

(1) **Structure**: A substrate manages a set of objects according to a structure. For example (Fig. 2), a table organizes cells into rows that the user interprets as records of a dataset and columns as values of a given attribute. Similarly, a diagram editor creates an ordered list of shapes that defines which object appears on top when they overlap.

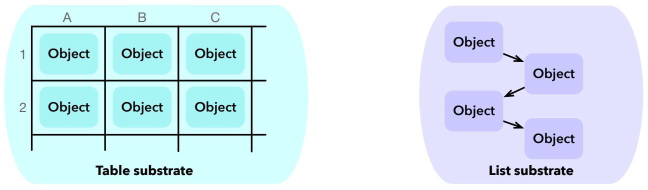


Figure 2: Substrates contain structured sets of objects: The Table substrate contains a spreadsheet with an object in each cell; the List substrate contains an ordered list of objects representing graphical shapes.

(2) **Constraints**: Substrates support explicit constraints that users can create, edit and remove. For example (Fig. 3), a spreadsheet might recompute formulas when the referenced cells change; a diagram editor might include alignment constraints that keep objects aligned when they are moved.

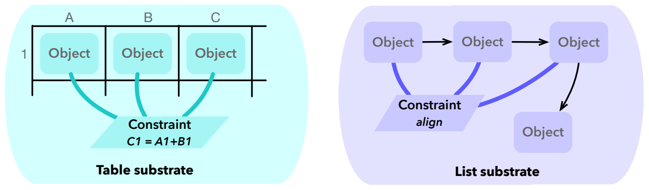


Figure 3: Constraints enforce relationships among objects in a substrate. The Table substrate contains a formula that calculates a sum; the List substrate contains a constraint that aligns the shapes.

(3) **Dependencies**: Substrates may depend on the content of other substrates or data from external sources and update their state whenever the source substrate or data changes. For example (Fig. 4), a graphing application may extract data from a spreadsheet substrate to create a plot that updates automatically whenever the spreadsheet changes. The spreadsheet itself may update according to new readings from an external sensor.

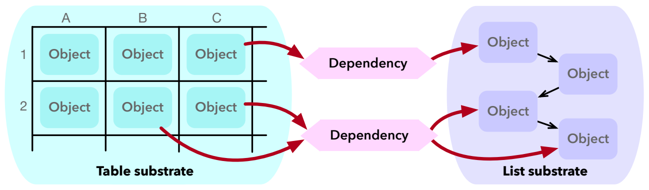


Figure 4: Dependencies enforce relationships among substrates. Shapes in the List substrate are tied to specified cells in the Table substrate.

3 Related work

We briefly review relevant previous work on conceptual models of interaction, on document-centric environments, and on improving power and simplicity in interactive systems. We then position our work with respect to previous uses of the word "substrate" in HCI.

3.1 Conceptual models

According to Norman [43], a conceptual model describes both the system designer's model of the system and the mental model users create as they interact with it. Unfortunately, we know of no formalisms that describe these models. Johnson & Henderson's conceptual model [28] is based on objects and operations but remains very high level. The principles of Direct Manipulation [51] — continuous representation of objects of interest, physical, incremental, reversible actions, rapid learning — are also high level and do not provide a framework for describing either the objects nor the actions. Similarly, Model-based user interfaces [42] are based on descriptions of tasks and "domain concepts" but do not provide specific guidance for how to describe or organize the concepts.

Jacob et al.'s reality-based interfaces [26] encourage exploiting the qualities of real-world physical objects in the digital world, but lack specific concepts. Ullmer et al. [57]'s Tokens+Constraints conceptual model for tangible interaction is more precise and closer to our work, but is mostly descriptive and lacks generative principles that can inform the design of novel interfaces.

We build upon two more relevant approaches: Höök & Löwgren's [24] "strong concepts" serve as intermediate-level knowledge that can inform new designs, whereas Generative Theories of Interaction [7] offer an actionable approach to creating and applying such concepts. Instrumental Interaction [4] is a key inspiration for the concept of a *substrate* and its design principles [8], together with our empirical studies of designers and end users. Although substrates complement instruments by describing the context in which they operate, the concept of substrate is independent from that of instrument and is applicable in non-instrumental interfaces.

3.2 Document-centric environments

The design of the first commercial graphics workstation, the Xerox Star [29], was centered on the concept of document and did not feature applications. Users could freely combine different types of content — text, images, tables, graphs — in the same document and interconnect them so that, for example, a bar chart would update automatically when the associated data table was edited.

Since then, digital environments have become overwhelmingly application centric, each specializing in a single content type. In the early 1990's, Microsoft's OLE[1] and Apple's OpenDoc [13] introduced document-centric models that let users include different types of content into their documents. Unfortunately, these systems did not push the document metaphor far enough, since each data type still required its own commands. Rather than creating a unified environment where different types of content and tools coexist and interoperate, they simply made application boundaries less visible.

Software suites such as Microsoft Office,[2] Adobe Creative Cloud[3] and Affinity[4] attempt to blur the application boundaries with, e.g., dynamic links that update automatically when importing content from another application in the suite, but they do not support tools that work across applications.

Our model is closer to the original document-centered approach: Substrates separate content from the commands used to manipulate them and promote a form of interoperability such that commands work for different types of substrates. Even so, substrates can also be used in traditional applications.

3.3 Power and simplicity in interactive systems

Designing interactive systems involves a trade-off between *power of expression* and *simplicity of execution* [38]. Most environments that offer users more power are programming environments aimed at developers rather than end-users. Alan Kay spoke of software as "clay" [31] and created malleable environments, most notably *Smalltalk* [20]. More recently, *Lively* [25] offers a similar approach in a web browser. User-driven development [34] specifically targets end users by letting them program new features themselves. Although one could envisage this type of programming environment for substrates, this is not the goal of our current work.

[1] https://learn.microsoft.com/en-us/cpp/mfc/ole-background?view=msvc-170
[2] https://www.microsoft365.com
[3] https://www.adobe.com/creativecloud.html
[4] https://affinity.serif.com

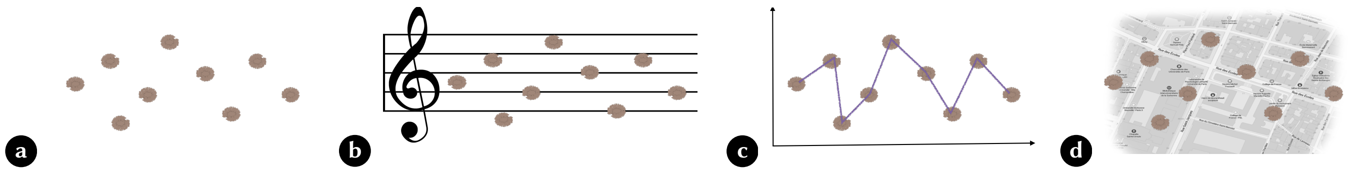


Figure 5: a Substrates provide both meaning and constraints to a set of dots. b A musical score transforms these dots into notes on a staff. c A graph constrains the dots to underlying data values. d A map specifies that dots correspond to addresses.

Instead, our work echoes the spirit of the *Alternate Reality Kit* [52], where objects appear to follow the laws of physics and can be manipulated in predictable ways with respect to other objects, thus encouraging exploration. Although we do not seek the same kind of literalism, our goal is similar, namely to facilitate the free combination of objects through polymorphic tools and substrates. A number of systems show the value of empowering users without sacrificing simplicity of interaction: *Buttons* [39] let users embody interactive behavior into sharable, configurable buttons; *Interface attachments* [44] augment existing applications based on their surface representations; and *Scotty* [16] injects code into existing applications to augment or replace their functionality. Our goal is similar — to encourage more open environments that can easily incorporate new functionality. However, rather than relying on ad hoc solutions, we define actionable, unifying principles that enable such diversity and flexibility.

3.4 Previous uses of the word "substrate"

We have already used the term "substrate" in previous work. *Paper substrates* [17] enable composers to create complex musical works by linking and layering sheets of interactive paper that represent and interpret digital data. For example, translucent substrates can reveal and interpret data from underlying substrates and users can simply draw a line to interconnect two substrates. Maudet et al. [40] highlight the difficulty graphic designers face when trying to represent their mental models of layouts in current software tools, and demonstrates how *Graphical substrates* can address this gap. *Narrative substrates* [21] capture selected traces of a player's activity in an online game and transform them into persistent, interactive content accessible to other players.

Webstrates [32] uses Web technologies to implement shareable dynamic media. The authors define "information substrates" as "*software artifacts that embody content, computation and interaction, effectively blurring the distinction between documents and applications*". The Webstrates server propagates changes in a web page's Document Object Model (DOM) to other clients in real time, and lets users *transclude* (include) one document within another. While this definition is similar to ours, a key difference is that Webstrates supports only one kind of substrate, the DOM, whereas our approach supports multiple levels of interconnected substrates.

In summary, our work extends and unifies previous work by proposing a more precise and operational definition of the concept of substrate. We build on Beaudouin-Lafon's definition [6]: "*A substrate is a digital computational medium that holds digital information, possibly created by another substrate, applies constraints and transformations to it, reacts to changes in both the information and the substrate, and generates information consumable by other substrates*". The next section provides in-depth descriptions of the characteristics of substrates.

4 Substrates

Our key insight is that focusing on objects of interest and commands is not sufficient to increase power and simplicity, we must also consider the *structure* that contains the objects of interest and the *effects* of commands on these objects. First, users infer the capabilities of an application not only from the objects they perceive, but also from their environment i.e. the *structure* that hosts them. Revealing the structure may therefore facilitate interaction. For example, seeing a grid implies that content belongs to cells and that the users can manipulate the rows and columns.

Second, users often apply and re-apply the same command to a set of objects in order to maintain a particular relationship among these objects. For example, users must manually re-align a set of aligned objects after moving one of them. Transferring the maintenance of the alignment to the system as a persistent constraint would increase the power of the interface.

Third, users should be able to tailor the environment to meet their needs. They may want to "tweak" constraints without breaking them, for example, to adjust the alignment of an odd-shaped logo in a diagram editor without losing the offset when the set of aligned objects is moved. They may also want to reuse content by creating templates with placeholders for new data, for example, to create a set of inter-related budget formulas in a spreadsheet and save them as a template for use when creating next year's budget.

4.1 Substrates structure the objects they contain

Substrates help both users and the system interpret the meaning and capabilities of the objects contained within the substrate. For example, if we look at a set of dots on the screen (Fig. 5a), it is not clear what they mean or what operations can be performed on them. However, if we clarify that the dots exist within a particular substrate (Fig. 5b,c,d), both their meaning and how they can be manipulated becomes clear.

In Figure 5a, the user expects to be able to move the dots around, without any particular constraint. In the context of a musical staff (Fig. 5b), the dots clearly represent the notes of a music score and the user expects that they can be moved only on or between the lines of the staff. In Figure 5c, the same dots are tied together by a line and the two axes convey the fact that they represent the data points of a line plot. Here, the interpretation is that the dots are tied to a data table and cannot be moved directly, but instead that the data table should be changed in order to change the graph. Finally, in Figure 5d, the dots are laid out on a map and are interpreted as locations. As with the music staff, the user expects to be able to move the dots only to positions that represent legal addresses. The user may also expect to be able to create new dots by entering an address, e.g. in a text field.

Each of these four substrates thus offer different meaning and capabilities for action. They help users understand the *affordances* [19] of the objects of interest (here, the dots). Note that appearances may be deceiving: a snapshot of a music staff does not afford the same interactions as an interactive music application. Note too that in addition to the perceived structure, users can also rely on available commands and tools to discover the substrate's capabilities.

4.2 Substrates manage constraints

Users dislike repetitive actions. For example, the grid layout of a table provides a structure that affords entering values into cells. In order to add together the numbers in a column, the designers could provide a command that lets users select a column of numbers and then apply the "Sum" command, which would enter the sum of the numbers in the cell below the column. The user would then have to re-select the cells and re-apply the "Sum" command each time they change a value in the column. This is similar to most current content-authoring applications: Diagram editors provide commands for aligning objects, but not for keeping them aligned. Some Word processors provide commands for counting the number of words in a selected piece of text, but do not update the word count as the user edits the text.

The fundamental difference between the table described above and a real spreadsheet is that the latter manages *persistent* relationships between cells: rather than having to add numbers each time they make a change, users simply specify that a cell must contain the sum of a set of cells, and the system maintains that constraint whenever a value changes. In other words, the power of a spreadsheet formula comes from replacing the *effect* of a command (adding a set of numbers) by a *persistent relationship* among these numbers. We call this process the *reification of an effect*. Reifying an effect is a form of transfer of responsibility from the user to the system, whereby the user expresses a relationship and lets the system maintain it.

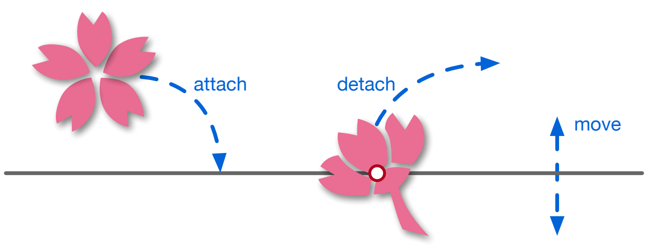


Figure 6: StickyLines: The user moves an object (blue dashed arrow) onto a StickyLine (horizontal line) to *attach* it and away from it to *detach* it. When attached, the constraint (red dot) ensures that the object *moves* (right blue dashed arrows) with the StickyLine.

Note that we are *not* suggesting that applications should support the kind of formulas found in spreadsheets. Formulas are good for spreadsheets because spreadsheets manipulate numbers, and math formulas provide a well-known way to define relationships among numbers. Reifying effects into constraints should be designed to be meaningful for the users in the target domain.

For example, designers of a diagram editor can reify visual relationships among objects into constraints such as alignment, inclusion or distribution. StickyLines [11] illustrate how an alignment constraint can be represented by a guideline to which objects are attached (Fig. 6). Dragging the guideline moves the attached objects. This contrasts with traditional alignment commands or the Alignment Stick [47], which let users align objects but do not retain the alignment. StickyLines have been shown to be up to 40% more efficient than traditional alignment commands both in time and number of user actions [11].

As another example, designers of a word processor can reify relationships among document elements such as words, paragraphs and sections. Textlets [22] illustrate how users can assign behaviors to parts of a text document, such as counting words or highlighting occurrences of a search pattern. Word counts and search occurrences are updated as the user edits the text (Fig. 7). This contrasts with traditional word processors, where users must re-issue the word-counting command every time the text changes and find occurrences one by one.

In summary, constraints bring power to substrates by letting users specify persistent relationships rather than having to maintain them themselves. These constraints in turn provide meaning to the substrate, by expressing design goals that users must otherwise manage in their heads [40].

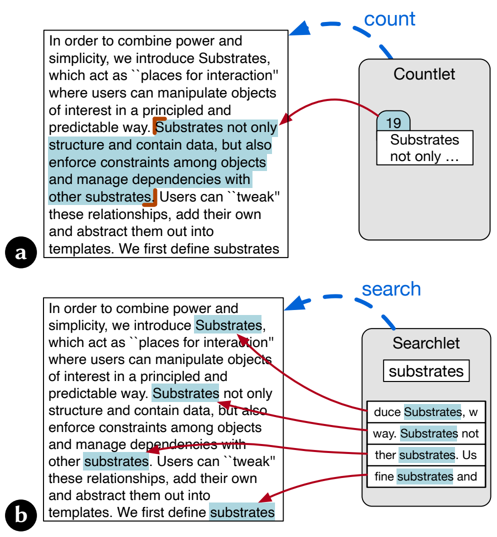
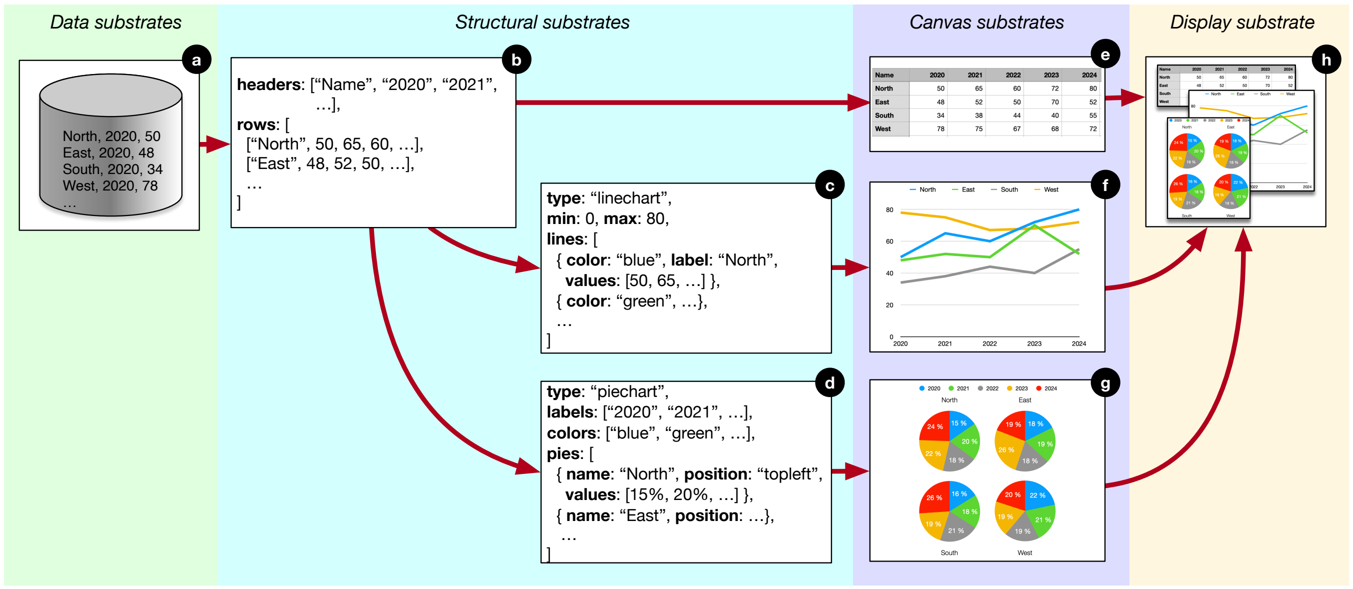

In order to combine power and simplicity, we introduce Substrates, which act as ``places for interaction'' where users can manipulate objects of interest in a principled and predictable way. Substrates not only structure and contain data, but also enforce constraints among objects and manage dependencies with other substrates. Users can ``tweak'' these relationships, add their own and abstract them out into templates. We first define substrates

Figure 7: Textlets: (a) A *countlet* is attached (blue dashed arrow) to the selected text, creating a constraint (red arrow) that displays the number of words in the text. (b) A *searchlet* for the word "substrates" is attached (blue dashed arrow) to the text; each occurrence creates a textlet.



```
North, 2020, 50
East, 2020, 48
South, 2020, 34
West, 2020, 78
…
```

```
headers: ["Name", "2020", "2021",
          …],
rows: [
  ["North", 50, 65, 60, …],
  ["East", 48, 52, 50, …],
  …
]
```

```
type: "linechart",
min: 0, max: 80,
lines: [
  { color: "blue", label: "North",
    values: [50, 65, …] },
  { color: "green", …},
  …
]
```

```
type: "piechart",
labels: ["2020", "2021", …],
colors: ["blue", "green", …],
pies: [
  { name: "North", position: "topleft",
    values: [15%, 20%, …] },
  { name: "East", position: …},
  …
]
```

Name	2020	2021	2022	2023	2024
North	50	65	60	72	80
East	48	52	50	70	52
South	34	38	44	40	55
West	78	75	67	68	72

Figure 8: a A data substrate provides input to b a structural substrate representing a table, which itself provides input to two other structural substrates representing a c line plot and a d pie chart. Each structural substrate is represented by a canvas substrate containing graphical shapes (e, f, g) which are finally rendered as pixels on the h display substrate. The red arrows represent dependencies where the head substrate reacts to changes in the tail substrate.

Interactive and polymorphic constraints. Once an effect has been reified into a constraint, making it *interactive* adds power. For example, spreadsheet formulas can refer to a *range* of cells by specifying its two ends, e.g., `A5:A10`. Ranges make formulas more powerful because the result is automatically recalculated when the user inserts or deletes rows within the range. Ranges can also be edited directly: Editing a formula in Microsoft Excel highlights the relevant ranges; The user can then adjust the range simply by dragging its handles.

Designers can turn the constraints that result from reifying effects into manipulable objects. For example, users can move a StickyLines guideline as if it were any other object. They can also add and remove objects from the alignment simply by dragging them next to or away from the guideline. In Textlets, the scope of a word-counting "countlet" can be adjusted by dragging its end handles. These interactions are so simple and obvious that they were inferred almost immediately by users, what psychologists call "one-trial learning".

Designers can also make more powerful *polymorphic*[5] constraints that work with multiple types of objects. For example, an alignment constraint can be applied to different shapes in a diagram editor, but also in other substrates such as the icons in a file manager, the windows on the desktop, or the text margins in a word processor. Turning constraints into objects and making them polymorphic encourages user exploration by letting them experiment with what works and what does not. This in turn helps them understand the underlying principles of the substrates and enables technical reasoning [45, 48].

[5]Polymorphism is one of the principles of Instrumental Interaction [8], defined as the ability for a tool to act upon objects of different kinds.

4.3 Substrates manage dependencies

Designers need to understand both how the underlying data structures affect how objects are presented to the user and the corresponding effect on how users interact with them. The substrates illustrated in Figure 5 are all visual: Each set of dots is represented by an ordered list within a *canvas substrate*. However, the musical score, the plot and the map in Figures 5b,c,d have a richer structure than just a set dots. We define *structural substrates* as non-visual substrates that represent these underlying structures. For example, a musical score substrate specifies the tones, a plot substrate describes the axes and the coordinates of each data plot, and a map substrate determines the location of each address. These structural substrates must then be linked to canvas substrates to produce visual representations.

Dependencies tie substrates together (Fig. 8). For example, the canvas substrate representing the line plot (Fig. 8f) is tied to the structural substrate describing the content of the plot (Fig. 8c). Structural substrates may themselves depend on other structural substrates. For example, the *plot* substrate (Fig. 8c) is tied to a *table* substrate (Fig. 8b) that holds the data being plotted. The table substrate, in turn, is tied to a *data substrate* (Fig. 8a), e.g., a database.

On the visual side, the content of the display is a grid of pixels that can be thought of as a *display substrate* (Fig. 8h) tied to the various canvas substrates (Fig. 8e,f,g). In practice, however, the display substrate is internal to the operating system and is not accessible as a full-fledged substrate. Designers may also create auditory or haptic substrates to support other modalities.

Substrates are *reactive*: they react to changes in the substrate(s) that they depend on. Figure 8 illustrates these dependencies and shows how multiple canvas substrates represent the same source data (Fig. 8a) as a table (Fig. 8b→e), a line graph (Fig. 8b→c→f), and a pie chart (Fig. 8b→d→g). A change in the data substrate updates the other substrates and, ultimately, the display.

Conversely, substrates must be able to *react to user input*: if a user tries to move a dot in the line plot, the substrate may ignore the change if the dot location is controlled by a dependency. Alternatively, it may interpret the user action as a change to the underlying substrate. For example, moving a dot in the line plot could update the data table but constrain the movement vertically. Moving a dot in the map of Figure 5d could snap it to the nearest address.

Finally, the organization of substrates into multiple levels opens up interesting collaboration possibilities. Sharing the data table substrate allows one user to view it as a line plot and another user as a pie chart, with both representations being updated as the data changes. Alternatively, they could share the canvas substrate and see and manipulate the same representation, as in WEBSTRATES [32].

In summary, dependencies add power by enabling complex combinations of substrates. They also bring simplicity, since changes propagate automatically, thus relieving users from duplicating them across multiple representations.

4.4 Substrates enable adjustments: tweaking

While constraints and dependencies provide power, they can also be ... constraining. Users often want to make specific adjustments to objects without losing the benefits of the underlying constraints. For example, presentation software includes master slides that specify the size and font of each text box. When a title box is too small for the intended text, the user typically adjusts the size or font to make it fit. This unfortunately breaks the master slide's constraint. A better solution would record that the constraint has been relaxed but is still in place if, for example, the title is shortened later.

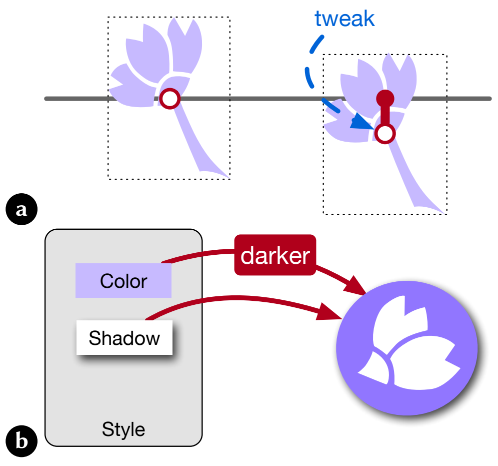


Figure 9: Tweaking: (a) The object's center point determines its location on the StickyLine (left). The user can offset the attachment point to improve the visual alignment (right). (b) Styles automatically copy their attributes to attached objects. The user can independently adjust these constraints to transform the copied value, here, to darken the color.

STICKYLINES introduced the notion of a "tweak" (Fig. 9a): When the user moves an object attached to a guideline with arrow keys instead of the mouse, the system records the offset as a reified object or "tweak". The tweak is attached to the object and is applied when moving the object to a different StickyLine. Styles could use a similar approach, so that the user could tweak the brightness of an object's color (Fig. 9b).

More generally, a tweak adjusts the value assigned to an object's attribute by a command or constraint. A tweak is persistent: it adjusts the value each time the attribute changes. Changing the style's color to green in Figure 9 would change the logo's background color to a darker green. Tweaks add power by letting users loosen constraints instead of reverting to manual control and losing the benefit of the constraint entirely.

4.5 Substrates enable specialization: templating

Users often want to customize their environment by creating specialized versions of the objects provided by an application. A common approach is to provide *templates* that can be filled out by the user. However, creating templates is often complex. Substrates support a simple way of defining templates: The user starts with an existing substrate, with its objects and constraints, and replaces some of the objects with placeholders. The resulting template can then be used as any other substrate. Figure 10 shows how a simple template can be created from an alignment guideline.

Spreadsheet users could create templates from a set of cells and formulas with placeholders for selected data. Each formula is itself a template because cell references are relative rather than absolute. This is part of the power of spreadsheets: formulas can be reused in different contexts and yet "do the right thing".

Transforming a collection of objects and constraints in a substrate into a template is a powerful way of abstracting behavior and is also simpler than directly creating the template from scratch, such as when defining master slides with presentation software. Substrates give users more flexibility, especially when combined with tweaking: A user can apply a template and tweak the resulting instance, yet keep the benefit of the persistent constraint between the template and the instance.

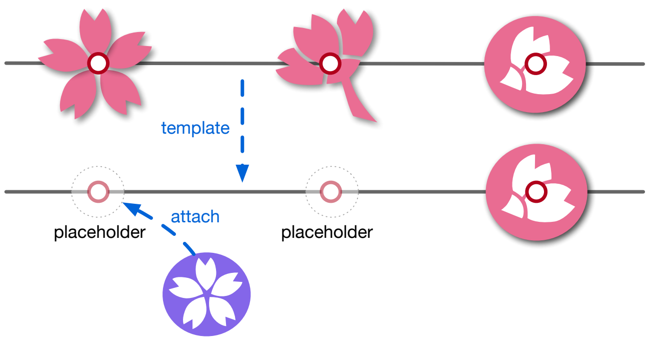


Figure 10: Templating: The user begins with a StickyLine with three attached logos (top) and creates a template by transforming the left two into placeholders but keeps the far right logo as is (bottom). A new logo can then be added.

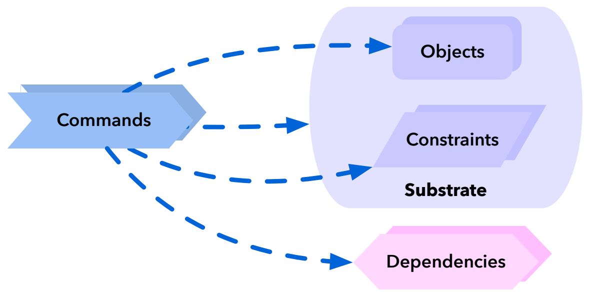


Figure 11: Commands create, delete or modify (blue dashed arrows) objects, constraints, dependencies and substrates.

4.6 Summary

Figure 11 summarizes the concepts involved in substrates. A *substrate* contains *objects* and *constraints* that apply to these objects, and maintains these constraints in response to changes from *commands*. Substrates can be linked to other substrates through *dependencies* that express how the state of the objects in the source substrate affects the objects in the target substrate. Dependencies are reactive and update the target objects' state whenever the corresponding source objects change.

5 Assessing Substrates using Generative Theory

One of the challenges in proposing a new conceptual model is how to justify and evaluate it. Running a controlled evaluation study, especially a quantitative one, on such a rich topic is impractical: because design is such a complex process, it would be difficult to attribute the differences observed between two designs — one with and the other without substrates — solely to the use of the concept. We thus use a two-pronged strategy that combines analytical and empirical assessments based on the Generative Theories of Interaction [7] approach. A Generative Theory of Interaction is grounded in theories of human behavior and operationalizes these theories into actionable concepts and principles. Researchers use the following lenses to apply these concepts and principles to existing systems and new design problems:

1. **Analyze:** Can existing systems be deconstructed in terms of the concepts and principles of the theory?
2. **Critique:** If they are present, do they improve or hinder the system? If they do not exist, what problems does this cause? Could applying the principles improve the system?
3. **Construct:** Can these concepts and principles inspire new ideas when designing a new system?

We found that the principles introduced by Instrumental Interaction [7, 8], namely Reification, Polymorphism and Reuse, apply to Substrates. Whereas Instrumental Interaction focuses on commands, reified as instruments, Substrates focus on the *effects* of these commands, reified into substrates and constraints.

Table 1 lists the original principles of Reification, Polymorphism and Reuse (blue boxes) from [8] as well as two new principles, Adjustment and Specialization (purple boxes), with brief definitions and examples for each. The left column refers to instruments that result from reifying a command, whereas the right column introduces substrates that result from reifying the *effect* of a command.

Both instruments and substrates can be polymorphic i.e. applied to objects of different types. The principle of Output Reuse introduced in Instrumental Interaction corresponds to reusing the objects, e.g. through copy-pasting. We extend it to the reuse of constraints. On the other hand, the new customization principles — Adjustment and Specialization — are also directly relevant to Instrumental Interaction. The previously mentioned principles of tweaking and templating apply to substrates. We next introduce two corresponding principles for instruments: tuning (adjusting a command) and currying (specializing an instrument).

The next subsections describes the theoretical underpinnings of Substrates, apply the analytical and critical lenses to existing commercial systems and research projects, and reports on applying the constructive lens in three HCI classes.

5.1 Theory: Affordances, Technical Reasoning, Naive Physics and Co-adaptation

Instrumental Interaction and Substrates are grounded in theories of human behavior, including *affordances* [19], *technical reasoning* [45, 46] and *co-adaptation* [14]. Gibson's theory of affordances supports our concept of substrate in that the context of an object affects its perception and therefore the perception of its affordances. This was illustrated in Figure 5 where the context of the dots sets the expectation for certain capabilities for interaction[6].

The combination of tools[7] and substrates further reinforces the perception of potential affordances. For example, a text entry tool evokes the ability to add text, which the user may not have perceived otherwise. Conversely, a substrate that contains text evokes the ability to edit this text, suggesting the existence of a text entry tool. If text is visible in the substrate but no text entry tool is present, the user may conclude that the text is not editable — maybe it is an image instead.

Technical reasoning focuses on human tool use and describes how humans' knowledge of "abstract technical principles" let them take advantage of object and tool properties to solve interaction problems. In the physical world, a flat, thin rigid object such as a knife can cut through softer objects. Our recent work has shown that technical reasoning is at play when interacting with digital content [48, 49], based on what we call "interaction knowledge", and that users can use digital tools in unusual ways to solve problems.

The use of tools, however, cannot be dissociated from the objects they interact with and their context, in that the operation of the tool may be constrained by the properties of the objects and of its surroundings. In other words, the substrate and the constraints it embodies affect the way tools work with the content. For example, in a word processor, text is organized into lines inside pages — a fundamental constraint of a text substrate. All text editing tools must work with these constraints. Conversely, the constraints imposed by a substrate suggest specific tools for managing them. In word processing, tools should let the user control text layout constraints, such as line spacing and margins.

[6]Note that these may be false affordances [18] if the capability is not actually available.
[7]Atau Tanaka [55] offers a nuanced distinction between musical instruments designed for creative expression and tools designed to accomplish tasks. For the purposes of this paper, we use the terms "instrument" and "tool" interchangeably.

Table 1: Expanded set of Instrumental Interaction principles. Original definitions are blue; new principles are purple.

	Command focus	**Effect focus**
Reification	Transforms a **command** into an **instrument** *Example:* "Fill" command becomes the paint bucket tool	Transforms command **effects** into a **substrate** *Example:* Calculation becomes a spreadsheet formula
	Instruments let users:	**Substrates let users:**
Polymorphism	Apply a **command** to **multiple types** of objects *Example:* Change color of shapes, text and backgrounds	Apply a **constraint** to **multiple types** of objects *Example:* Align shapes, text, images and windows
Reuse	Apply **previous commands** to objects *Example:* Execute a macro	Apply **previous effects** to objects *Example:* Copy-paste objects and/or constraints
Specialization	**Curry** a parameter **value** to create a **new instrument** *Example:* Create personalized format brushes	Create a **template** from **objects and constraints** *Example:* Create a master slide with placeholders
Adjustment	**Tune** command parameter **values** *Example:* Click-drag to resize while creating a rectangle	**Tweak** command **effects** *Example:* Offset a logo from a guideline

Technical reasoning is strongly related to diSessa's notion of "naive physics" [15]. A key difference is that naive physics emphasizes the fact that the "laws" that are inferred are not necessarily the actual laws of physics. Indeed, users often make incorrect assumptions about what is and is not possible with digital content. This is often due to the differences in how applications deal with similar content, making it difficult to infer general "laws of interaction". The concept of naive physics therefore encourages designers to create substrates and tools with consistent behaviors that users can easily understand. Designers should also consider extending existing substrates with new types of constraints rather than creating new substrates from scratch.

Finally, the importance of customization is inspired by the phenomenon of co-adaptation [14], which describes how animals both adapt to their environment but also actively adapt it for their own purposes. Mackay [35] explores this phenomenon with respect to human users and describes the related phenomenon of *reciprocal co-adaptation* [7] where human users and intelligent agents both learn (or adapt to) and modify (or adapt) each other's behavior. Co-adaptation describes the process by which users learn rules to "master" the predicted use of the system, but also take advantage of those rules to create custom solutions for their particular needs. This implies that the underlying interactive systems, in this case substrates, need predictable rules and behaviors that enable user customization.

In summary, the theories of affordances and technical reasoning strongly support the relevance of substrates as a new concept, while naive physics suggests that substrates should be generic and extensible, and co-adaptation justifies the need for supporting customization. These theories also emphasize the duality between tools and substrates, which we see as a promising avenue for design.

5.2 Analysis and critique of existing systems

Content substrates. We can analyze existing content-authoring applications in terms of instruments and substrates. The work area where users create content is a substrate: it contains the objects that make up the content and the application imposes constraints on them, such as the layout of a text document into lines and pages in a word processor, the front-to-back ordering of shapes in a diagram editor, or the stacking of layers in an image-editing application.

However, these substrates are relatively limited and applying *effect reification* could greatly improve their power. We see examples of effect reification in research projects such as Object-Oriented Drawing [58], where the user can create constraints linking the values of object attributes so that, for example, they have the same color. Para [27] lets users create a collection of shapes and vary parameters along the collection. The relationships are maintained and/or adjusted as the user makes changes, such as mapping the size of the shapes to their rank in the collection.

Existing applications would also benefit from unifying the constraints they manage. For example, Microsoft Word features different commands for numbering lists, sections, figures and references, each with their own controls. Textlets [22] address this problem by reifying numbering into number and reference templates, which both unify and generalize the concept of numbering.

Inspectors and styles as substrates. Inspectors can be viewed as substrates that show a target object's attributes as editable values. However, the link between the attributes and the object may not be clear. Inspectors can set command parameters such as the brush color, change attributes of selected objects such as their text size, or change attributes of a style, which in turn affect various objects in the content.

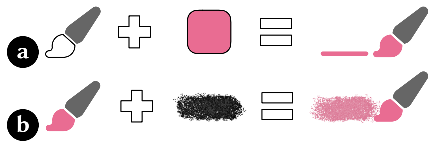


Figure 12: Currying a command. A brush tool requires two parameters: paint color and brush shape. By currying the (a) color and then (b) brush shape, the user can create more specialized brushes to suit their needs.

Rather than using a single inspector for different purposes, which causes confusion and increases complexity, the user should be able to create multiple inspectors that target different objects, similar to the attribute objects in Object-Oriented Drawing [58]. A similar problem occurs with the search-and-replace command, which performs only one search at a time. By reifying the search command into a command and a substrate, Searchlets [22] let users manage multiple search-and-replace tasks in parallel.

Structural substrates. Content-authoring applications feature a structural substrate that underlies the visual representation of the content. It can be a simple list, such as the list of layers in an image-editing application or the list of slides in a presentation application, or a more complex structure, such as representing a text layout or 3D model. Different visualizations may show different aspects of the structure: Users can rearrange slides in a light table or reorder layers in a "layers" panel. However, as with inspectors, current applications usually feature a single alternative visualization. What if the user could create several light tables to test alternative presentation orders for different audiences? Or compare the effects of different layer orders in a diagram editor?

Interacting with content through multiple views synchronized through a shared structural substrate provides users with greater power of expression. For example, a word processor's outline facilitates (re)organizing content. Histomages [10] lets users modify images through pixel-based manipulations of an image's histogram, enabling otherwise tedious transformations such as changing the sky's color. Similarly, a diagram editor could provide both a classic substrate that manages geometric shapes and a substrate that manages the corresponding planar map [3], which would be updated dynamically when moving the shapes in the canvas substrate [2]. Other examples of multiple views include WritLarge [59] and its different levels of interpretation of pen strokes, SenseCape [53] and its canvas and hierarchy views for sensemaking, Sketch-n-Sketch [23] and its code and output views, and visualization applications such as VegaLite [50] or StructGraphics [56] where data, its visualization and their mapping are all separately editable.

Adjustment: Tweaking Constraints and Tuning Commands. Tweaking involves overriding or offsetting a constraint in order to make a small adjustment without losing the benefit of the original, persistent constraint. In current systems, the only solution is to remove the constraint altogether. For example, when applying a style to an object, if one of the attributes of the object is changed, the link to the style, at least for that attribute, is lost. Tweaking lets users make small changes such as offsetting an aligned object or making a color a bit brighter (Fig. 9), so that this change will be applied when the base color of the style is changed.

Interestingly, the concept of adjustment can also be applied to the instrument side of Table 1. We define *Tuning* as the ability to adjust the parameters of a command as it is issued. For example, users can interactively define the size of a shape while creating it. Similarly, the user could select a color and drag the mouse to interactively adjust its brightness. Unlike a tweak, this adjustment does not persist — neither the instrument nor the object remember the adjustment that was applied, the object simply receives the modified color.

Specialization: Creating templates and currying commands. Templates exist in a number of applications, but with limited capability. In many cases, templates can only be created for an entire document, not as a set of objects, constraints and placeholders that can be used in different places within the document. Presentation applications use templates (or slide masters) that can be applied to individual slides, but constrain the set of placeholders to specific items such as the title and body text. Our approach instead lets users capture a set of objects and constraints into a template that can be reused with different objects. For example, a user might create a two-column slide in a presentation application and then extract a template containing just the layout information for use with other content.

Templates support customization by letting users create specialized substrates. As with Adjustment, the Specialization principle can also be applied to the instrument side of Table 1. Applying a command often requires specifying parameters. For example, when picking a paintbrush tool, the user must also select the color and brush shape. Paintbrush tools remember the last parameters used, but if the user wants to alternate between two or more brush shapes and colors, they must re-specify them each time they pick up the tool. Having access to the "large blue brush" and the "thin red brush" would be simpler and more efficient.

The concept of *currying* from functional programming helps us achieve this effect. Currying *specializes* a function by fixing one of its parameters. Here, it lets users create specialized copies of a command by fixing one or more parameters. Figure 12 shows the use of currying to first specialize the color of the paintbrush tool and then its brush shape.

Users can thus create the set of commands and tools that fit their needs and adjust this set as they go. Except for the interface of the reMarkable tablets[8], which features two pen tools that can be configured independently, we are not aware of any system that supports currying as a general principle.

In summary, on the command side, users should be able to create personalized tools based on existing ones. On the substrate side, they should be able to create their own substrates to fit their needs. The former increases simplicity while the latter increases power.

User interface substrates. Beyond the content itself, applications may include other substrates. The interface of the application, which organizes the menus, toolbars, inspectors, panels and other floating windows, can be seen as a substrate, designed for controlling the application's content. This substrate is typically very rigid,

[8]https://remarkable.com

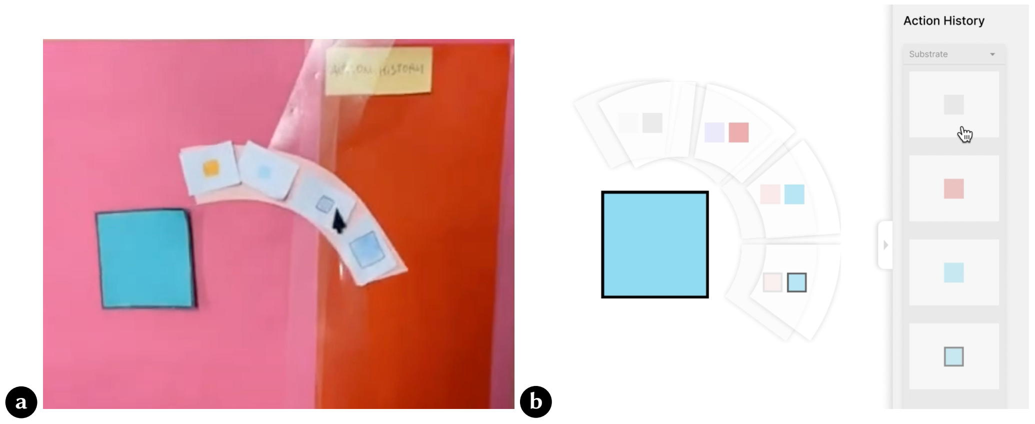


Figure 13: a Students create low-resolution paper prototypes and b high-resolution video prototypes. This user sees her previous style modifications to the blue square reified into a semi-circular clipboard substrate that is being dragged to an "action history" panel. The hi-res version shows how previous style modification sequences can be copied, tweaked and reused.

which is surprising considering how important it is for creative professionals to organize their physical space according to their needs. Users of content-authoring applications can sometimes rearrange the panels, but rarely their content, and even more rarely create their own toolbars and panels.

Similarly, the organization of windows on the screen is left to the window manager, with little control given to the user beyond moving and resizing windows. A window substrate would let users organize them more freely, and include operations such as tabbing, turning and snapping [5] or laying them out in 3D [9]. Treating windows as rectangular shapes would let users apply constraints, such as alignment, and commands, such as zooming, thus bringing standard features from diagram editors to window management.

This analysis shows that, although many existing systems include some characteristics of substrates, most lack a coherent set of rules that help users understand how to interact with them. We argue that identifying the basic characteristics of a substrate and corresponding actionable principles for creating them can help designers increase the power and simplicity of both existing and new interactive systems.

5.3 (Re)-Constructing Google Slides

We have taught the theory of Instrumental Interaction over the past decade, but have only recently begun using the Generative Theory approach with modules on substrates, tweaking and currying. This section describes our experiences teaching these new concepts to graduate HCI students.

Teaching Approach. We included substrates in two seven-week graduate-level HCI classes (approximately 32 students per year) at our university and in a one-day Master Class with 20 participants from three other universities. Each class begins with a description of the Generative Theory approach, followed by lectures on the concepts of instruments and substrates. We also ask students to engage in various exercises to help them to understand related theories — affordances, technical reasoning and co-adaptation — with both physical and digital examples.

We explore the Google Slides' interface, chosen for its familiarity and ease of access. Google Slides poses many interesting design challenges: Students can consider individual slide creation, animation within or across slides, one- or two-dimensional views of groups of slides, slide masters (or templates), as well as different modes for creating, sharing, editing or presenting slides. After demonstrating how to analyze and critique Google Slides, we illustrate how to use the principles to brainstorm new forms of interaction that offer more power, with greater simplicity. Next, students form groups and select a particular aspect of Google Slides to redesign by applying the concepts of instruments and substrates and the associated principles. We ask them to first use Google Slides and identify a feature or interaction that annoys them[9]. Finally, we ask students to create a new design concept and, in the two seven-week courses, illustrate it with a series of low-fidelity paper-based video prototypes [37] that show how the concept addresses each of the principles. After we critique their designs, they revise their concepts and create either a high-fidelity video prototype or a working demonstration. We upload selected projects into our Interaction Museum website[10].

Results. The following examples, drawn from student projects in 2023 and 2024, illustrate how students applied the concepts of instruments and substrates to the re-design of Google Slides.

History – One project explored how to reifiy localized histories of the user's previous commands. Figure 13 shows a low-resolution paper prototype and a high-resolution video prototype. The system captures in-context local histories of the user's previous commands with respect to a particular graphical object. The user hovers the cursor to pop up a semi-circular list of these changes, which can be moved to an "action history" panel. Users can also copy, delete and rearrange the commands to create re-usable macros. Both the commands and the panel can be tweaked and the collapsed set

[9]Interestingly, this is very difficult for many HCI students, who are so accustomed to traditional menu-and-button interfaces that they do not notice when the interaction is cumbersome or confusing.
[10]http://interaction.museum. See the Appendix for an example of how one group systematically illustrated their design.

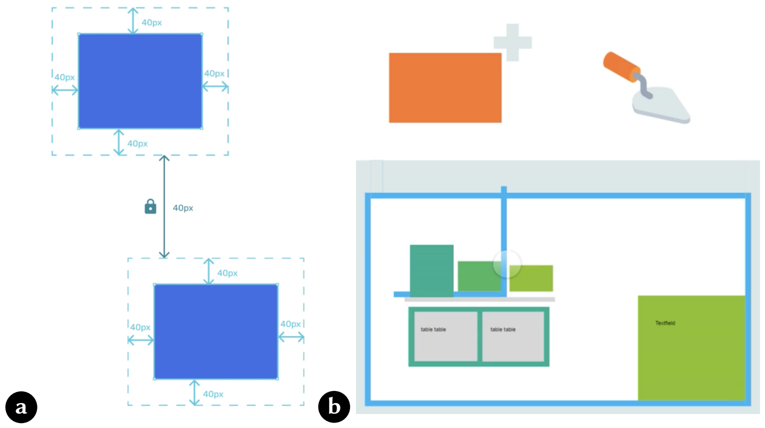


Figure 14: The user can (a) reify, tweak and reuse the space around objects and distances between margins, and (b) reify the margins in the layout grid as manipulable objects.

of style modifications can be treated as a visible, interactive style command that can be applied to other objects.

Compared to GOOGLE SLIDES, this project both increases the functionality available to the user (more power), while offering an easy-to-use interaction style (greater simplicity). Instead of GOOGLE SLIDES' limited version history, it offers a personalized, detailed set of recent user actions that can be easily reused.

Margins – Two groups considered the problem of managing margins and distances between objects in a slide. One group reified the margins and their measurements surrounding each graphical object, creating interactive substrates that can be copied, pasted and tweaked, as well as saved as a sort of reusable "margin style" (Fig. 14a). Another group reified the margins in a grid layout, transforming the negative space between the slide content into a kind of flexible "cement" that preserves the inter-object relationships even as the content changes. This serves as a flexible, interactive substrate that applies polymorphically to text, shapes and images, and supports user-defined constraints that dictate how the objects fit together. In both cases, users can preserve and reuse tweaks that offset specialized content from the margin and curry personalized tools that create different kinds of constraints.

Compared to GOOGLE SLIDES, both projects offer greater power of expression by allowing users to define their own reusable margin styles or margin constraints. Both offer more precise control of the layout than GOOGLE SLIDES, yet provide a simpler, more direct form of interaction with the margins, instead of having to find the appropriate pull-down menu and fill in points or percentages of line height in a dialog box.

Color – Another group created more sophisticated, interactive and reusable color palettes (Fig. 15). Users can select a sequence of colors in the color space of their choice, which is then reified into a path that retains their relative distances within the underlying color space. Users can move either points or the whole path, or transfer a path into another color space. Their concept is explicitly designed to support color tweaking. For example, paths should be tweaked when the color space changes from purple to yellow, to add more contrast. Users can reify and tweak any color relationship or attribute, e.g. hue, saturation, transparency, and apply those tweaks to any colored object in the canvas. Users can also visualize current and previous color paths to explore the color space.

GOOGLE SLIDES provides either a palette of selected colors or a dialog box that accepts RGB numbers or a cursor to pick a particular hue. Unlike this project, users cannot specify relationships among colors, nor make fine-grained tweaks of those relationships. The students' color project offers much greater power, with a much greater set of functions, but the interaction itself is actually more direct and simpler for the user to control.

Translation – Another group explored how to add AI-based translation to GOOGLE SLIDES. After initially getting stuck and reverting to menu-based interaction, one student suddenly realized that she could create a "translation brush" that could be loaded with a particular language and brushed onto any text. From there, the group explored how to create language-based templates where users could create place-holders for text-based content and then generate new slides with the appropriate translations. They considered how to tweak specific translations and rearrange the space to accommodate, for example, different sizes (German takes more space than English) and different orientations (from English to Chinese, Hebrew or Persian). Thinking in terms of instruments and substrates helped them offer users a far more powerful set of translation capabilities, while creating a much simpler form of interaction.

GOOGLE SLIDES does not currently support automated text translation and only lets users turn spell checking on or off. This project lets users not only specify the scope of the text, but also the target language. The same direct interaction with a "translation" brush would work equally well for spell- and grammar-checking, offering a simpler yet more powerful approach to all three.

Other projects – Students generated a variety of other interesting concepts, including: reifying animation paths that can be applied polymorphically within and across slides; redesigning presentation mode to handle both live and recorded interaction with a timeline substrate; and reifying a "jump" from one part of a presentation to another to create interactive shortcuts. In each case, the students significantly improved the interaction to GOOGLE SLIDES through the

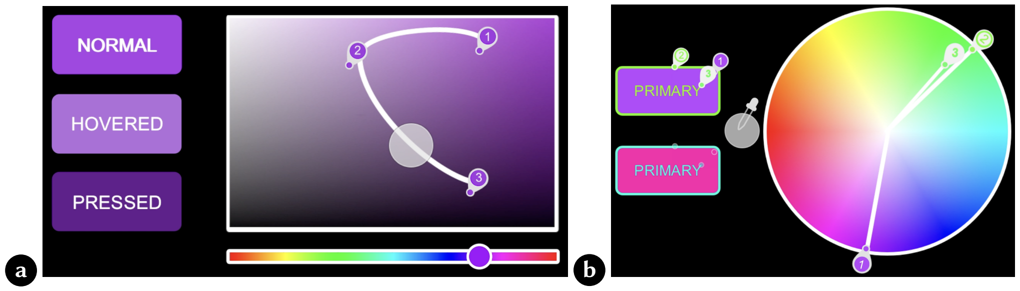


Figure 15: The high-res video prototypes show how the user can (a) draw a path through a color space where each point in the path is dynamically linked to three buttons (*normal, hovered* and *pressed*. Each color space has different characteristics, so the path can be tweaked if, for example, the colors shift to shades of yellow. The user can also (b) save select specific points and relationships in a different color space, and use them to define relationships between image elements.

use of substrates and instruments, with more powerful interfaces that offer users added functionality while preserving or increasing the simplicity of interaction with that functionality.

The inclusion of substrates, tweaking and currying to the most recent (2023 and 2024) versions of the course produced more innovative projects that were explored in greater depth than in previous years. The Masters students' course evaluations showed that the course was challenging but the majority said they "learned alot": "We learned very useful principles that we can use in the future for designing". One student said: "The tools are really powerful when used correctly." and another: "... loved creating one single tool and applying generative theory of HCI." 86% of the students said they "learned new design principles" and 76% said they "expected to use them in the future".

Although projects tended to focus on either instruments or substrates, all but one of 16 group projects[11] successfully generated examples of all ten principles presented in Figure 1. Applying these principles sparked many ideas and generated a great deal of excitement as they explored how to add both power and simplicity to GOOGLE SLIDES. Several students have successfully transformed their projects into Masters and Ph.D. theses as well as several recent HCI publications.

6 Discussion

6.1 Power and simplicity

We are interested in supporting experts, not just novices, and in providing an incrementally learnable path from novice to expert. While there are inherent trade-offs between power and simplicity, we have observed that the reification of commands into tools tends to increase simplicity, while the reification of effects into constraints tends to increase power. A path from novice to expert can be created by first offering tools and substrates with limited capabilities and then expanding them. For example, a paint tool could be curried to offer a set of paintbrushes with basic colors and a standard brush shape, then the user could be given access to the full paint tool with its inspector. Similarly, an alignment guideline could be made available first in its simplest form, then a more advanced version could provide additional capabilities, such as tweaking or distribution.

Instruments and substrates also offer users flexibility as they shift the focus of their tasks. Mackay's comparative analysis of interaction techniques [36] showed that, given an identical task and identical numbers of actions, the users' preferred interaction technique varies according to their intent — in that case, copying vs. creating something new — and a corresponding difference in which tools were preferred and more efficient. By giving users more diverse types of tools, but also letting them transfer some of their agency to the system with constraints, users gain new ways of adapting their practice to the task at hand. However, beyond the classical measures of time and error, we need additional objective measures of both power and simplicity in order to operationalize these differences into the design of controlled experiments. These measures should target not only short-term use, but also long-term use such as learning and recall.

6.2 Limitations and directions for research

Evaluating a conceptual model is challenging. While our experience using and teaching these concepts over the past few years has convinced us of its power, we still need better ways of assessing the added value of applying these concepts to real-world design problems, in both the short and long term.

Another challenge is turning designs based on these concepts into working software. Existing user interface toolkits and frameworks such as JAVAFX,[12] REACT[13] or QT[14] are based on programming patterns that do not support substrates: They decouple tools from substrates and do not support the reactive behavior of constraints and dependencies. Future work should create dedicated software toolkits that facilitate implementation of systems that embody the concepts presented in this paper. Over time, we hope that a set of "standard" substrates will emerge to facilitate further development.

Finally, even though we developed substrates in the context of graphical user interfaces, we believe that the concept can be applied, with adaptations, to other forms of interaction, including

[11] from the seven-week HCI classes

[12] https://openjfx.io
[13] https://react.dev
[14] https://www.qt.io

Virtual/Augmented Reality, tangible interaction, speech-based and multi-modal interaction. For example, Reactile [54] uses tangible interaction to program a swarm of robots with constructs similar to substrates and constraints.

7 Summary and Conclusion

Interactive systems face a fundamental trade-off between simplicity of interaction and power of expression. Our key insight is to focus on the *context* in which users manipulate objects of interest and capture it with the concept of *Substrate*. Substrates provide an environment that not only contains the user's objects of interest, but also manages constraints among these objects and dependencies with other substrates. Constraints and dependencies transfer the burden of maintaining invariants within and across substrates from the user to the system.

We also introduce two new generative principles to support customization — Adjustment and Specialization — and show that they apply to substrates as well as instruments. Adjustment includes *tuning* commands to dynamically modify parameters while applying them and *tweaking* constraints to make small but persistent offsets. Specialization includes *currying* commands to fix their parameters and creating *templates* from substrates.

This paper offers four key contributions: First, we introduce the concept of *substrate* and its key components: *objects*, *constraints* and *dependencies*. Second, we introduce two new design principles, *Adjustment* and *Specialization*, that enable customization. Third, we provide in-depth descriptions, with examples, of the characteristics of substrates. Finally, we illustrate how to meet the challenge raised in [7] by demonstrating how to apply the analytical, critical and constructive lenses of a Generative Theory of Interaction to assess the value of the concept of substrates and its associated principles.

In the future, we hope that these ideas will contribute to the creation of rich and versatile "places for interaction" where digital tools and digital content are designed to be as comprehensible, simple and powerful as tools and materials in the physical world.

Our long-term goal is to create a foundation for a "digital physics of interaction", similar to diSessa's "naive physics" but for the digital world, where users create mental models based on a coherent set of underlying mechanisms that dictate how to interact with digital material. We hope this work will encourage further exploration of this design space.

Acknowledgments

We thank the many students and colleagues who contributed ideas and discussions over the years, especially: Jessalyn (Sally) Alvina, Matthew Beaudouin-Lafon, Susanne Bødker, Marianela Ciolfi Felice, Jérémie Garcia, Camille Gobert, Jun Kato, Germàn Léiva, Carla Griggio, Viktor Gustaffson, Clemens Klokmose, Joe Malloch, Nolwenn Maudet, Joanna McGrenere, Arvind Satyanarayan, Theophanis (Fanis) Tsandilas and Anastasiya Zakreuskaya.

This work was partially supported by European Research Council (ERC) Advanced Grants № 321135 "CREATIV: Creating Co-Adaptive Human-Computer Partnerships" (P.I. Wendy Mackay) and № 695464 "ONE: Unified Principles of Interaction" (P.I. Michel Beaudouin-Lafon) under the European Union's Framework 7 and Horizon 2020 research and innovation programmes.

References

[1] Caroline Appert, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2005. Context matters: Evaluating Interaction Techniques with the CIS Model. In *People and Computers XVIII — Design for Life*, Sally Fincher, Panos Markopoulos, David Moore, and Roy Ruddle (Eds.). Springer London, London, 279–295.

[2] Paul Asente, Mike Schuster, and Teri Pettit. 2007. Dynamic planar map illustration. *ACM Trans. Graph.* 26, 3 (jul 2007), 30–es. https://doi.org/10.1145/1276377.1276415

[3] P. Baudelaire and M. Gangnet. 1989. Planar maps: an interaction paradigm for graphic design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '89)*. ACM, New York, NY, USA, 313–318. https://doi.org/10.1145/67449.67511

[4] Michel Beaudouin-Lafon. 2000. Instrumental Interaction: An Interaction Model for Designing post-WIMP User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands) *(CHI '00)*. ACM, New York, NY, USA, 446–453. https://doi.org/10.1145/332040.332473

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

[6] Michel Beaudouin-Lafon. 2017. Towards Unified Principles of Interaction. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter* (Cagliari, Italy) *(CHItaly '17)*. ACM, New York, NY, USA, Article 1, 2 pages. https://doi.org/10.1145/3125571.3125602

[7] Michel Beaudouin-Lafon, Susanne Bødker, and Wendy E Mackay. 2021. Generative theories of interaction. *ACM Transactions on Computer-Human Interaction (TOCHI)* 28, 6 (2021), 1–54.

[8] Michel Beaudouin-Lafon and Wendy E. Mackay. 2000. Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Palermo, Italy) *(AVI '00)*. ACM, New York, NY, USA, 102–109. https://doi.org/10.1145/345513.345267

[9] Olivier Chapuis and Nicolas Roussel. 2005. Metisse is not a 3D desktop!. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology* (Seattle, WA, USA) *(UIST '05)*. ACM, New York, NY, USA, 13–22. https://doi.org/10.1145/1095034.1095038

[10] Fanny Chevalier, Pierre Dragicevic, and Christophe Hurter. 2012. Histomages: fully synchronized views for image editing. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (Cambridge, Massachusetts, USA) *(UIST '12)*. ACM, New York, NY, USA, 281–286. https://doi.org/10.1145/2380116.2380152

[11] Marianela Ciolfi Felice, Nolwenn Maudet, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2016. Beyond Snapping: Persistent, Tweakable Alignment and Distribution with StickyLines. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. ACM, New York, NY, USA, 133–144. https://doi.org/10.1145/2984511.2984577

[12] Andy Cockburn, Carl Gutwin, Joey Scarr, and Sylvain Malacria. 2014. Supporting Novice to Expert Transitions in User Interfaces. *ACM Comput. Surv.* 47, 2, Article 31 (Nov. 2014), 36 pages. https://doi.org/10.1145/2659796

[13] Dave Curbow and Elizabeth Dykstra-Erickson. 1997. Designing the OpenDoc Human Interface. In *Proceedings of the 2nd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques* (Amsterdam, The Netherlands) *(DIS '97)*. ACM, New York, NY, USA, 83–95. https://doi.org/10.1145/263552.263581

[14] Charles C. Darwin. 1859. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* John Murray., Albemarle Street, London, England.

[15] Andrea A. diSessa. 1993. Toward an Epistemology of Physics. *Cognition and Instruction* 10 (1993), 105–225. https://api.semanticscholar.org/CorpusID:120960205

[16] James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2011. Cracking the Cocoa Nut: User Interface Programming at Runtime. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. ACM, New York, NY, USA, 225–234. https://doi.org/10.1145/2047196.2047226

[17] Jérémie Garcia, Theophanis Tsandilas, Carlos Agon, and Wendy Mackay. 2012. Interactive paper substrates to support musical creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. ACM, New York, NY, USA, 1825–1828. https://doi.org/10.1145/2207676.2208316

[18] William W. Gaver. 1991. Technology affordances. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New Orleans, Louisiana, USA) *(CHI '91)*. ACM, New York, NY, USA, 79–84. https://doi.org/10.1145/108844.108856

[19] James J. Gibson. 1979. *The ecological approach to visual perception.* Houghton, Mifflin and Company, Boston, Massachusetts.

[20] Adele Goldberg and David Robson. 1983. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[21] Viktor Gustafsson, Benjamin Holme, and Wendy E. Mackay. 2020. Narrative Substrates: Reifying and Managing Emergent Narratives in Persistent Game Worlds. In *Proceedings of the 15th International Conference on the Foundations of Digital Games* (Bugibba, Malta) *(FDG '20)*. ACM, New York, NY, USA, Article 46, 12 pages. https://doi.org/10.1145/3402942.3403015

[22] Han L. Han, Miguel A. Renom, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2020. Textlets: Supporting Constraints and Consistency in Text Documents. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376804

[23] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-Directed Programming for SVG. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. ACM, New York, NY, USA, 281–292. https://doi.org/10.1145/3332165.3347925

[24] Kristina Höök and Jonas Löwgren. 2012. Strong Concepts: Intermediate-level Knowledge in Interaction Design Research. *ACM Trans. Comput.-Hum. Interact.* 19, 3, Article 23 (Oct. 2012), 18 pages. https://doi.org/10.1145/2362364.2362371

[25] Daniel Ingalls, Tim Felgentreff, Robert Hirschfeld, Robert Krahn, Jens Lincke, Marko Röder, Antero Taivalsaari, and Tommi Mikkonen. 2016. A World of Active Objects for Work and Play: The First Ten Years of Lively. In *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Amsterdam, Netherlands) *(Onward! 2016)*. ACM, New York, NY, USA, 238–249. https://doi.org/10.1145/2986012.2986029

[26] Robert J.K. Jacob, Audrey Girouard, Leanne M. Hirshfield, Michael S. Horn, Orit Shaer, Erin Treacy Solovey, and Jamie Zigelbaum. 2008. Reality-based Interaction: A Framework for post-WIMP Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) *(CHI '08)*. ACM, New York, NY, USA, 201–210. https://doi.org/10.1145/1357054.1357089

[27] Jennifer Jacobs, Sumit Gogia, Radomír Mundefinedch, and Joel R. Brandt. 2017. Supporting Expressive Procedural Art Creation through Direct Manipulation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 6330–6341. https://doi.org/10.1145/3025453.3025927

[28] Jeff Johnson and Austin Henderson. 2011. *Conceptual Models: Core to Good Design.* Morgan & Claypool, San Rafael, California. https://books.google.fr/books?id=SfMkFC8jHdwC

[29] Jeff Johnson, Teresa L. Roberts, William Verplank, David Canfield Smith, Charles H. Irby, Marian Beard, and Kevin Mackey. 1989. The Xerox Star: A Retrospective. *Computer* 22, 9 (1989), 11–26.

[30] Alan C. Kay. 1977. Microelectronics and the Personal Computer. *Scientific American* 237, 3 (1977), 230–245. http://www.jstor.org/stable/24920330

[31] Alan C. Kay. 1984. Computer Software. *Scientific American* 3, 251 (1984), 52–59. https://doi.org/10.2307/24920344

[32] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology* (Charlotte, NC, USA) *(UIST '15)*. ACM, New York, NY, USA, 280–290. https://doi.org/10.1145/2807442.2807446

[33] Jingyi Li, Eric Rawn, Jacob Ritchie, Jasper Tran O'Leary, and Sean Follmer. 2023. Beyond the Artifact: Power as a Lens for Creativity Support Tools. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23)*. ACM, New York, NY, USA, Article 47, 15 pages. https://doi.org/10.1145/3586183.3606831

[34] Henry Lieberman, Fabio Paternò, and Volker Wulf (Eds.). 2006. *End User Development.* Springer Netherlands. https://doi.org/10.1007/1-4020-5386-X_1

[35] Wendy Mackay. 2000. Responding to cognitive overload : Co-adaptation between users and technology. *Intellectica* 30, 1 (2000), 177–193. https://doi.org/10.3406/intel.2000.1597

[36] Wendy E. Mackay. 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* (Trento, Italy) *(AVI '02)*. ACM, New York, NY, USA, 203–208. https://doi.org/10.1145/1556262.1556294

[37] Wendy E. Mackay. 2023. *DOIT: The Design of Interactive Things: CHI'23 Preview.* Inria, Paris, France. 88 pages.

[38] Wendy E. Mackay and Michel Beaudouin-Lafon. 2005. Generative Approaches to Simplicity in Design. In *International Forum: Less is More - Simple Computing in an Age of Complexity.* Microsoft Research, Cambridge, UK, 3 pages.

[39] Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. 1990. User-tailorable Systems: Pressing the Issues with Buttons. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Washington, USA) *(CHI '90)*. ACM, New York, NY, USA, 175–182. https://doi.org/10.1145/97243.97271

[40] Nolwenn Maudet, Ghita Jalal, Philip Tchernavskij, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2017. Beyond Grids: Interactive Graphical Substrates to Structure Digital Layout. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 5053–5064. https://doi.org/10.1145/3025453.3025718

[41] Joanna McGrenere and Gale Moore. 2000. Are we all in the same "bloat"?. In *Proceedings of Graphics Interface 2000 (GI'2000).* CHCCS/SCDHM, Montréal, Canada, 187–196. https://doi.org/10.20380/GI2000.25

[42] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. 2011. Past, Present, and Future of Model-Based User Interface Development. *i-com* 10 (11 2011), 2–11. https://doi.org/10.1524/icom.2011.0026

[43] Don Norman. 1998. *The Design of Everyday Things.* Doubleday, New York, NY, USA.

[44] Dan R. Olsen, Jr., Scott E. Hudson, Thom Verratti, Jeremy M. Heiner, and Matt Phelps. 1999. Implementing Interface Attachments Based on Surface Representations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. ACM, New York, NY, USA, 191–198. https://doi.org/10.1145/302979.303038

[45] François Osiurak. 2014. What Neuropsychology Tells us About Human Tool Use? The Four Constraints Theory (4CT): Mechanics, Space, Time, and Effort. *Neuropsychology Review* 24, 2 (2014), 88–115. https://doi.org/10.1007/s11065-014-9260-y

[46] François Osiurak, Christophe Jarry, and Didier Le Gall. 2010. Grasping the affordances, understanding the reasoning. Toward a dialectical theory of human tool use. *Psychological Review* 117, 2 (2010), 517–540. https://halshs.archives-ouvertes.fr/halshs-00485348

[47] Roope Raisamo and Kari-Jouko Räihä. 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* (Seattle, Washington, USA) *(UIST '96)*. ACM, New York, NY, USA, 157–164. https://doi.org/10.1145/237091.237113

[48] Miguel A. Renom, Baptiste Caramiaux, and Michel Beaudouin-Lafon. 2022. Exploring Technical Reasoning in Digital Tool Use. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. ACM, New York, NY, USA, Article 579, 17 pages. https://doi.org/10.1145/3491102.3501877

[49] Miguel A. Renom, Baptiste Caramiaux, and Michel Beaudouin-Lafon. 2023. Interaction Knowledge: Understanding the 'Mechanics' of Digital Tools. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. ACM, New York, NY, USA, Article 403, 14 pages. https://doi.org/10.1145/3544548.3581246

[50] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (jan 2017), 341–350. https://doi.org/10.1109/TVCG.2016.2599030

[51] Ben Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (Aug. 1983), 57–69. https://doi.org/10.1109/MC.1983.1654471

[52] Randall B. Smith. 1987. Experiences with the Alternate Reality Kit: An Example of the Tension Between Literalism and Magic. In *Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface* (Toronto, Ontario, Canada) *(CHI '87)*. ACM, New York, NY, USA, 61–67. https://doi.org/10.1145/29933.30861

[53] Sangho Suh, Bryan Min, Srishti Palani, and Haijun Xia. 2023. Sensecape: Enabling Multilevel Exploration and Sensemaking with Large Language Models. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23)*. ACM, New York, NY, USA, Article 1, 18 pages. https://doi.org/10.1145/3586183.3606756

[54] Ryo Suzuki, Jun Kato, Mark D. Gross, and Tom Yeh. 2018. Reactile: Programming Swarm User Interfaces through Direct Physical Manipulation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173773

[55] Atau Tanaka. 2011. *Sensor-based musical instruments and interactive music.* Oxford Academid, Oxford, UK, Chapter 12, 233–257. https://doi.org/10.1093/oxfordhb/9780199792030.013.0012

[56] Theophanis Tsandilas. 2021. StructGraphics: Flexible Visualization Design through Data-Agnostic and Reusable Graphical Structures. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2021), 315–325. https://doi.org/10.1109/TVCG.2020.3030476

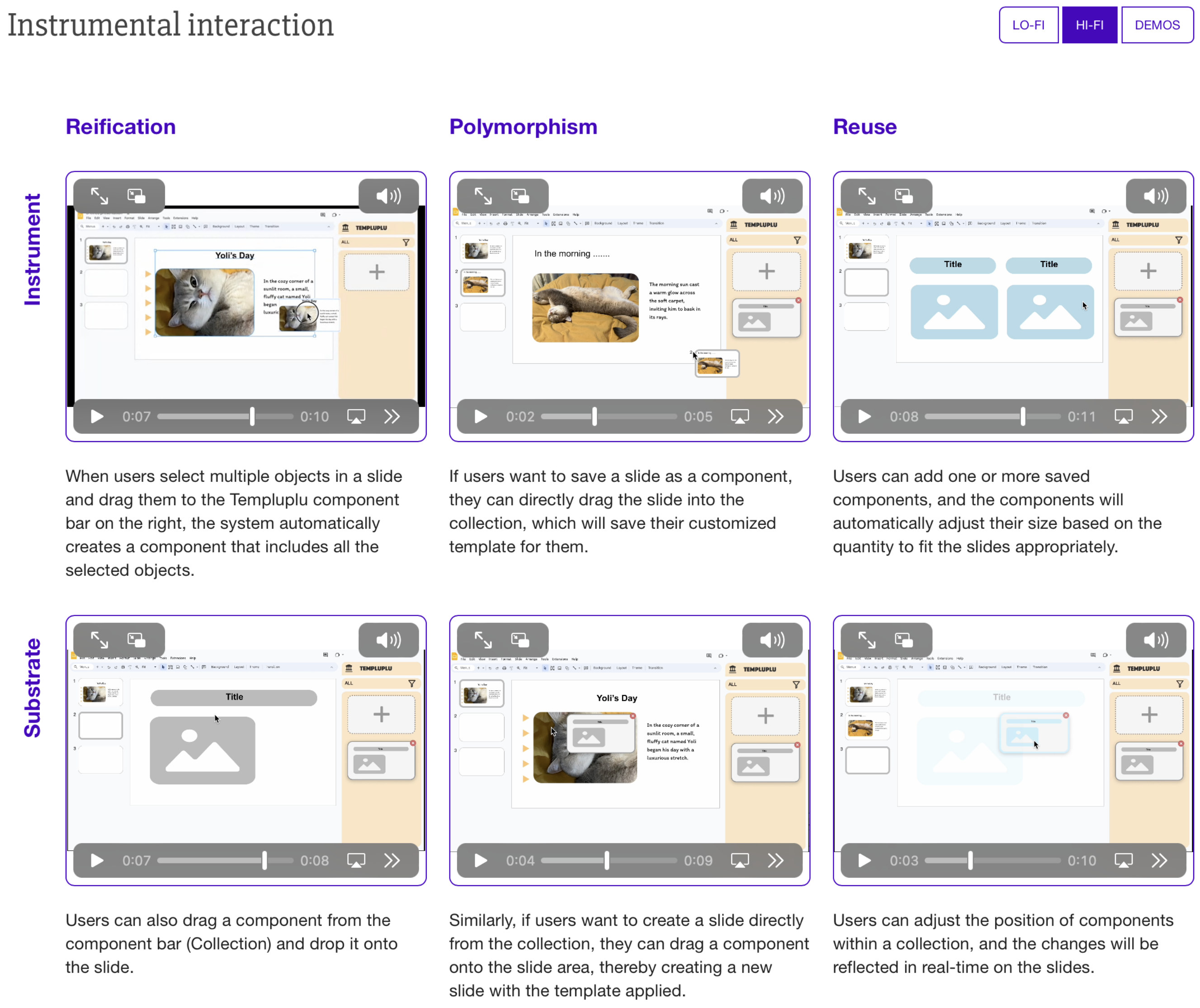
[57] Brygg Ullmer, Hiroshi Ishii, and Robert J. K. Jacob. 2005. Token+Constraint Systems for Tangible Interaction with Digital Information. *ACM Trans. Computer-Human Interaction* 12, 1 (March 2005), 81–118. https://doi.org/10.1145/1057237.1057242

[58] Haijun Xia, Bruno Araujo, Tovi Grossman, and Daniel Wigdor. 2016. Object-Oriented Drawing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. ACM, New York, NY, USA, 4610–4621. https://doi.org/10.1145/2858036.2858075

[59] Haijun Xia, Ken Hinckley, Michel Pahud, Xiao Tu, and Bill Buxton. 2017. WritLarge: Ink Unleashed by Unified Scope, Action, & Zoom. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 3227–3240. https://doi.org/10.1145/3025453.3025664

Appendix

The Interaction Museum (http://interaction.museum) offers a collection of both published interaction techniques from the research literature as well as examples of proposed interaction designs from our students at the Université Paris-Saclay (Fig. 16).



Instrumental interaction

Reification

When users select multiple objects in a slide and drag them to the Templuplu component bar on the right, the system automatically creates a component that includes all the selected objects.

Users can also drag a component from the component bar (Collection) and drop it onto the slide.

Polymorphism

If users want to save a slide as a component, they can directly drag the slide into the collection, which will save their customized template for them.

Similarly, if users want to create a slide directly from the collection, they can drag a component onto the slide area, thereby creating a new slide with the template applied.

Reuse

Users can add one or more saved components, and the components will automatically adjust their size based on the quantity to fit the slides appropriately.

Users can adjust the position of components within a collection, and the changes will be reflected in real-time on the slides.

Figure 16: The *Interaction Museum* website includes both low- and high-resolution video prototypes, as well as demos. This example shows six high-resolution video prototypes that illustrate the principles of reification, polymorphism and reuse with respect to instruments and substrates. (The currying and tweaking examples appear on a different page.)