Direct Manipulation Through Surrogate Objects

Bum chul Kwon, Waqas Javed, Niklas Elmqvist, and Ji Soo Yi
Purdue University
West Lafayette, IN, USA
{kwonb, wjaved, elm, yij}@purdue.edu

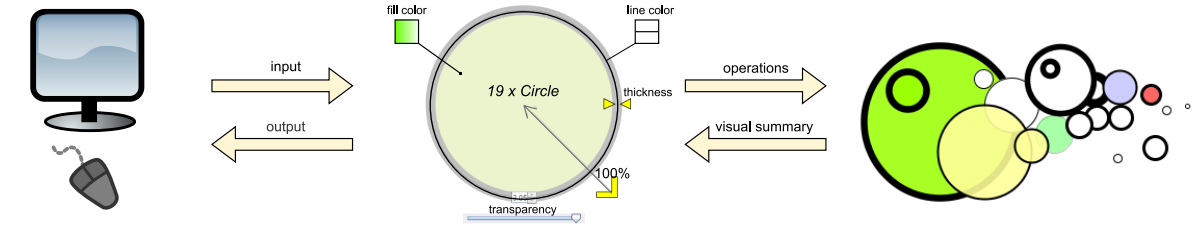

Figure 1. Conceptual diagram of Surrogate Interaction for a 2D vector editor. All interaction with the surrogate is forwarded to the shapes.

ABSTRACT

Direct manipulation has had major influence on interface design since it was proposed by Shneiderman in 1982. Although directness generally benefits users, direct manipulation also has weaknesses. In some cases, such as when a user needs to manipulate small, attribute-rich objects or multiple objects simultaneously, indirect manipulation may be more efficient at the cost of directness or intuitiveness of the interaction. Several techniques have been developed over the years to address these issues, but these are all isolated and limited efforts with no coherent underlying principle. We propose the notion of *Surrogate Interaction* that ties together a large subset of these techniques through the use of a surrogate object that allow users to interact with the surrogate instead of the domain object. We believe that formalizing this family of interaction techniques will provide an additional and powerful interface design alternative for interaction designers, as well as uncover opportunities for future research.

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces—*Interaction styles*; I.3.6 Computer Graphics: Methodology and Techniques—*Interaction techniques*

Author Keywords

Direct manipulation, instrumental interaction, design.

General Terms

Design, Theory

INTRODUCTION

For well over two decades, the direct manipulation paradigm [25, 26] has held sway in interface design, prompting designers to minimize indirection and to transform domain objects themselves into first-class interfaces. Despite being criticized for its deficiencies in managing abstract, group-based, or meta-level operations [12], challenged by agent-based and conversational approaches [17, 27], and extended with new measures for compatibility (similar to Norman's naturalness of mapping [18]) and integration of the input device, direct manipulation remains one of the basic tenets of user interface design to this day [28].

Although many novel approaches to resolve the weaknesses of the direct manipulation model have been proposed over the years, these tend to be isolated measures that do not seem to fit within a coherent framework. However, in this paper, we identify a subset of these techniques that share salient properties in how abstract, group-based, and meta-level operations are managed. We call this family of techniques *Surrogate Interaction* because they introduce *surrogate objects* that users can interact with instead of the real domain objects, and where changes are then propagated to the linked objects (Figure 1). While this increases indirection—albeit in a disciplined fashion—where direct manipulation stipulates decreasing it, this approach supports control over attribute-rich domain objects that would be difficult to access using direct manipulation, seamless and visible manipulation of multiple objects simultaneously, and access to abstract, intangible, meta-level properties and domain objects.

Consider a 2D vector editor such as Adobe Illustrator or the Open Source program Inkscape: graphical applications with multiple attribute-rich domain objects, i.e., geometric shapes. The standard solution is to provide multiple dialogs, control panels, and context menus to change graphical properties such as the line width, stippling, stroke color, fill, tex-

ture, transparency, and gradient of the shapes, only leaving a few attributes such as position, size, and rotation controlled by direct manipulation. Using the surrogate interaction approach, a 2D vector editor provides a single surrogate of the currently selected shape—or shapes—augmented with direct manipulation controls to change these graphical attributes. The surrogate looks like the selected shape and propagates all interaction to it, yet is larger, located in a well-defined place (a side panel or context menu), and can be annotated without interfering with the image being edited.

More specifically, by providing a surrogate of a domain object in a well-defined area of the screen, we can solve the access problem—the surrogate can be large and, by virtue of being a meta representation, is not required to convey information. It is thus easy to acquire and can be decorated with the necessary interface widgets. Furthermore, the surrogate also clearly shows that actions on it may affect multiple domain objects by indicating the selected objects, and its visual representation may unify the common attributes of the selection. Finally, surrogates can also be created for intangible properties, such as layout, ordering, or grid spacing.

In the following section, we give the background on direct manipulation and some of its weaknesses. We then derive the surrogate interaction framework from both new and existing academic and commercial applications that exhibit these commonalities. We go on to present a set of design examples for multidimensional visualization, vector drawing, and mobile devices for embodied interaction. We conclude with implications and constraints for user interface design.

BACKGROUND

Direct Manipulation

Introduced by Ben Shneiderman in 1982 [25, 26], the *direct manipulation* paradigm has held sway over user interface design for more than two decades. Before direct manipulation, the interaction between human and computer mostly relied on a conversation metaphor (e.g., a command-line interface), which is relatively indirect and abstract. As a backlash against this, direct manipulation promoted a more explicit way of interacting with computers. The paradigm is characterized by four main principles [26]:

1. Continuous representation of the object of interest;
2. Physical actions instead of complex syntax;
3. Rapid, incremental, and reversible operations whose impact on the object of interest is immediately visible; and
4. Layered or spiral approach to learning that permits usage with minimal knowledge.

These principles have helped make interaction more natural, intuitive, and predictable, thus causing applications to become easier and more efficient to learn and use.

However, direct manipulation did not win this reputation easily. Since its introduction, many researchers have criticized the paradigm [5, 7, 11, 12]. Among these, Frohlich [11, 12] provided a comprehensive review of the evolution and debate around direct manipulation. He argued that direct manipulation is not a panacea, and it actually has only limited utility because it is not abstract enough to fully express the user's intended interaction. More specifically, he listed seven tasks that are challenging to direction manipulation [12]—see Table 1. Based on these challenges, he predicted that the use of manipulation-based interaction would be limited and constrained. He further claimed that, as an alternative, conversation-based interaction (e.g., software agents) would complement manipulation-based interaction because it could provide users with more abstract and expressive interaction. The debate continued through the discussion between Shneiderman and Maes in 1997 [27].

In spite of (or thanks to) strong criticism, direct manipulation has evolved and appears to have stood the test of time. Numerous examples of academic research and commercial products explicitly and implicitly adopt the direct manipulation paradigm in their interaction design. The influences of direct manipulation are not limited to traditional WIMP (windows, icons, menus, pointer) interfaces, but are widely applied to other types of interfaces—often known as post-WIMP [35] or reality-based interaction [15]—such as information visualization (e.g., [37]), augmented/virtual reality (e.g., [22]) and direct touch interfaces (e.g., [23]).

Other researchers have expanded and refined the idea of direct manipulation, the most notable being the *instrumental interaction* model proposed by Michel Beaudouin-Lafon [3, 4]. The main idea of this model is to separate out the concept of an *instrument* as the mediator between users and domain objects in the interaction between human and computer. Instruments are first-class objects (e.g., a scroll bar and a paint brush) that can be used in many different contexts (e.g., a scrollbar that can operate on both a word processing document and a drawing canvas). Beaudouin-Lafon also introduced three properties to evaluate these instruments [3]:

1. *Degree of indirection:* a two-dimensional measure of the spatial and temporal distance introduced by instruments;
2. *Degree of integration:* the ratio between the degrees of freedom of the instrument and the input device; and
3. *Degree of compatibility:* a measure of similarity between actions on the instrument and the feedback on the object.

Given these properties, Beaudouin-Lafon argues that an instrument with low indirectness, high integration, and high compatibility will be a good instrument. These evaluation criteria could guide and generate the design for techniques and applications. Recently, Klokmose and Beaudouin-Lafon [16] further expanded the model by proposing the Views, Instruments, Governors and Objects (VIGO) architecture that allows many users to interact with multiple devices through diverse interfaces and on multiple surfaces.

We revisited the challenging tasks for direct manipulation that Frohlich listed [12]—see Table 1—and found that many of these challenges have been overcome through several variations of direct manipulation. When there are no proper visual representations of an object, visual representations are often created (or reified) to allow users to directly manipulate the object. For example, in order to allow a user to repeat

	Challenging Tasks	Solutions
1	Referring to previous actions	Interaction histories
2	Scheduling actions	Event scheduling
3	Identifying unseen objects	Shaking, jittering
4	Identifying groups of objects	Multi-object selection
5	Performing repetitive actions	Visual macros
6	Performing concurrent actions	Multitouch interaction
7	Specifying precise actions	Zooming

Table 1. Seven challenges for direct manipulation (adapted from [12]).

a certain action, visual macros (e.g., Apple Mac OS X Automator and Yahoo Pipes [38]) have been developed. When there already exists proper visual representations, other interaction techniques have been invented to overcome the identified issues (e.g., zooming to show the details of domain object). Furthermore, there still exists many examples of conversation-based interaction designs (e.g., the `cron` Unix command for scheduling repeated tasks) as well as compromises between conversation-based design and manipulation-based design (e.g., the Windows XP Scheduling Wizard). However, in general, direct manipulation seems to have tackled most of these challenges and has proliferated in interaction design for the last decades.

Challenges of Direct Manipulation

However, not all of these challenges have been effectively overcome for direct manipulation. Some of the limitations that Frohlich mentioned in 1993 still exist in current direct manipulation applications. In this work, we focus on the following issues (corresponding to challenges 7, 6, and 3):

- *Access:* Manipulating small, distant, or attribute-rich objects under limited space, high density, or high precision;
- *Multiple objects:* Manipulating multiple objects simultaneously as a group (including group attributes); and
- *Intangible properties:* Manipulating intangible object properties (abstract properties with no visual form).

The first of these challenges—**access**—is based on the fact that manipulating small objects can be cumbersome with direct manipulation. For example, when an object in a vector drawing application becomes too small, it may be difficult to select the object and to grab the handle instrument on the object in order to resize it. This problem is compounded if there are many objects in the same area of the screen, if the objects are located in a distant part of the screen, or even outside the screen in ubiquitous computing. Some of these issues can be partially solved by zooming the view for more detail and precision or other distortion techniques, but these approaches incur additional effort and cognitive load.

Another aspect of the same problem is manifested if the objects are rich in attributes. There may not be enough visual space to expose interface controls to manipulate all attributes, such as for a transistor object in an electric circuit CAD application. Alternatively, such controls detract from the visual appearance of the objects, such as information-carrying marks in a visualization, where adding such controls to the display would affect the visual representation.

The second problem we address is that interacting with **multiple objects** is often difficult using direct manipulation. When a user selects multiple domain objects with varying values (e.g., text with different font sizes), the aggregated properties of all selected objects typically cannot be properly presented, so the visual representation becomes ambiguous.

In addition, interacting with multiple objects using direct manipulation is often not nearly as intuitive as interacting with a single object. For example, suppose that a group of objects are selected, and the user resizes one of the objects in the group. How should the other objects in the group react to this user manipulation? One popular solution used in many applications, such as Microsoft PowerPoint, is that the single object that the user manipulates serves as a representative for the whole group, so the other objects in the group are also influenced by the same manipulation. This approach saves effort because it allows the user to resize all of the objects in the group by resizing a single representative. However, the approach could be criticized due to, again, its ambiguity. What if the selected objects have different shapes and sizes? Suppose that the user wants to increase the width of every object by 10 pixels—will resizing the representative object by 10 pixels cause the other objects to increase their widths by the same amount, or will their widths be increased proportionally? Much of the predictability of the direct manipulation interface has been lost in this situation.

Finally, the third problem we address deals with manipulating **intangible object properties**—abstract properties that have no natural visual form. For example, the spacing between multiple objects is not easily presented as a domain object, so it is difficult to allow direct manipulation of this attribute. Similar examples of intangible properties could be sorting order, layout, and object alignment. Many of these intangible properties arise from the multiple object challenge discussed earlier—these are attributes that do not exist other than for a group of objects, not for individual objects.

Currently, these intangible properties are manipulated in a relatively indirect manner. For example, Adobe Illustrator provides a toolbox for rearranging objects, allowing a user to choose between seven different layout types. Although the user might see the effects of layout changes immediately, there is significant spatial and cognitive distances between the domain objects and the instruments.

Data visualization gives rise to another type of intangible properties: summary statistics. These properties are especially challenging to turn into visible elements because they also do not belong to a single object but only become meaningful when multiple objects are related. Since these summary statistics might introduce visual clutter as well, it is often challenging to include them on the display.

The common denominator among the above problems is that they stem from the main strength of direct manipulation—the directness of the interaction (i.e., the low degree of indirection [3]). Attempting to resolve these issues would mean relaxing this central design principle. In the next sections, we will study how to do this in a disciplined fashion.

EVIDENCE OF SURROGATE INTERACTION

This paper proposes surrogate interaction as a unifying model for improving access, multiplicity, and intangibility in direct manipulation by introducing a small degree of indirectness between the user and the domain objects through surrogate objects. Below we review existing tools, both academic and commercial, for evidence of surrogate interaction.

3D Object Manipulation

Manipulating 3D objects is a difficult task, and different approaches have been designed to aid or solve it. Some of these draw on a surrogate interaction model to ease the interaction. Go-go interaction [22] deploys a surrogate hand to manipulate distant objects. World-In-Miniature [30] (WIM) are small-scale copies of the full 3D environment that the user is currently inhabiting. The miniature is linked to a physical plane the user is holding—yielding an out-of-body third-person view of the world—and rotating or moving the plane will affect the miniature world accordingly. Notably, the WIM itself is a first-class object, allowing the user to interact directly with objects visible in the miniature (but which may not be visible to the user in their first-person view of the world), thus causing the environment to be updated.

Even more relevant to surrogate interaction are the voodoo dolls proposed by Pierce et al. [21]: miniatures of actual first-class 3D objects in the world. Any interaction performed on the voodoo doll is immediately propagated to the real object it is linked to, and visual feedback is reflected on the doll to indicate state changes. However, the concept does not support manipulating more than one object at a time. The Monkey [9] (and commercial Monkey 2) device is essentially a physical Voodoo doll: a biped armature that can be used to control a virtual 3D character on screen.

Figure 2. Paper doll interface in *World of Warcraft*. Changes in the doll (surrogate) will immediately affect the character (domain object).

Computer Games

Elements of surrogate interaction also exists in a number of computer games. For example, a common design mechanism, particularly for role-playing games (RPGs) where the player's character is wearing all manners of armor, weapons, and jewelry, is the *paper doll interface*. A paper doll is simply a miniature version of the player character where the user can access different equipment slots to dress the character in items from the inventory (just like a real paper doll). It is also often combined with information about the character's abilities, health, and bonuses (*World of Warcraft* in Figure 2).

The paper doll interface is clearly an example of surrogate interaction, where the doll is the surrogate for the player character (the domain object). Any changes to the equipment on the doll will be immediately reflected in the appearance of the character. More specifically, the mechanism solves the access problem—exposing all of these equipment slots on the 3D player character itself would cause visual clutter, and the paper doll is an elegant way to avoid this.

Interactive Legends

Legends have long been used as visual dictionaries in cartography, and this idea has also been extended to other applications—Edward Tufte [33] strongly advocate their use, and exemplify them, among other ways, through surrogate representations of people in photographs using silhouette legends [34]. With the advent of computerized maps, cartographers have begun to propose *active legends* that change color encodings as the user interacts with them [20].

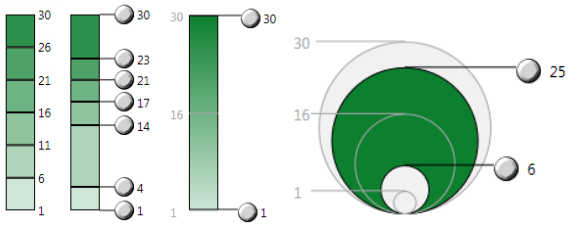

Figure 3. Interactive legends for filtering in visualization [24].

This trend has also carried across into visualization, prompting *interactive legends* as input devices that both save display space and are intrinsically integrated into the visual representation [32]. In particular, Riche et al. [24] evaluated the use of interactive legends for filtering and selection in multidimensional visualization (Figure 3), and found that they yield faster perception of data values. Similar to scented widgets [36], which incorporate visual displays into standard widgets, these interactive legends serve as surrogate objects representing visual elements in the visual representation, directly selected and filtered using the surrogate object.

Drawing Editors

Graphical editors are examples of interaction-heavy applications with attribute-rich domain objects (2D shapes). Perhaps for this reason, some editors already employ surrogate interaction concepts. Microsoft Office 2007 uses graphical depictions of what a selected shape will look like if different styles are applied. Inspectors in OmniGraffle [31] (Figure 4) are small windows containing visual surrogates that convey the state of the selection. Similarly, Apple's Keynote [2] incorporates inspectors for direct manipulation of shapes and slides, such as the Graphic Inspector (Figure 5).

In all of these examples, the surrogates are (to lesser or greater extent) visually reminiscient of the linked domain object, they summarize the state of the domain object, and they can be interacted with using direct manipulation, thus causing updates to propagate to the linked domain objects.

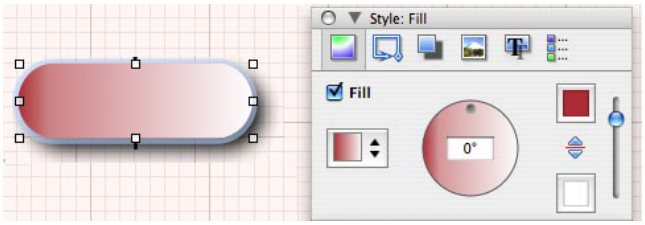

Figure 4. Style inspector in OmniGraffle Professional [31] for the Mac.

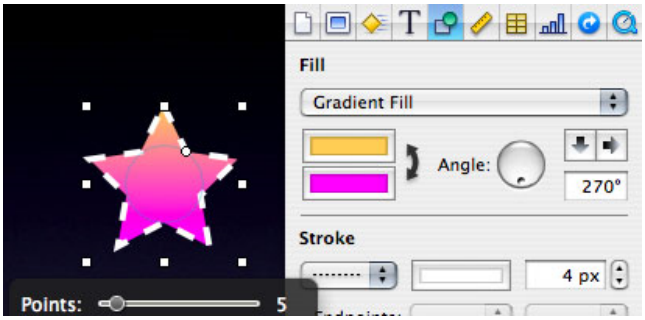

Figure 5. Graphic inspector in Apple Keynote [2].

SURROGATE INTERACTION

Instead of letting users control attributes of a domain object by interacting directly with its visual representation—as stipulated by both direct manipulation and instrumental interaction—surrogate interaction provides a *surrogate object* to interact with. The surrogate is an archetype of the domain objects, and any changes done to the surrogate will immediately affect the domain objects themselves.

More specifically, drawing on evidence in existing academic and commercial applications, we define *surrogate interaction* by means of the following characteristics:

1. Existence of surrogate objects that are visually reminiscient of the domain object/objects being manipulated;
2. Direct manipulation of object/meta attributes on the surrogate object that are propagated to the domain object; and
3. Visual summaries of the domain object state.

Consider a vector drawing editor where users can draw, move, and edit shapes by clicking and dragging with their pointer directly on the canvas using direct manipulation. Supporting surrogate interaction here would imply introducing a surrogate representation of a selected shape on the drawing canvas. This surrogate would allow users to change line style, width, and color as well as the fill color and style. Exposing control of these attributes in a direct manipulation fashion on the canvas itself is not practical due to space and layout constraints (not to mention being detrimental to the appearance of the image that the user is drawing), and would have had to be done using dialog boxes or context menus.

In the terms of the instrumental interaction model, introducing a surrogate object (or meta-object) such as this will increase the degree of spatial indirection in the interface. However, we are still obeying the design principles of direct manipulation by providing continuous visual representations of the domain objects with physical actions that are incremental, reversible, and provide immediate visual feedback. Surrogate interaction also solves the access, multiple object, and intangible property problems outlined earlier in this paper. In the following subsections, we shall see how.

Manipulating Single Objects

As we have seen, accessing domain objects using direct manipulation may be difficult if the visual space is limited or the number of attributes associated with the domain object is high. Furthermore, maintaining a high degree of compatibility between action and effect may be impossible in limited visual space. For surrogate interaction, we sidestep the access problem by allocating a part of the display outside of the main window to the visual representation of a large-scale surrogate of the domain object. Even though we now have introduced an additional level of spatial indirection, the surrogate is large and therefore easy to acquire and manipulate.

Having a surrogate representation for the domain object provides a number of additional benefits. Most importantly, by virtue of having more display space available in the surrogate than for the actual domain objects, we are able to integrate many more user interface components for controlling the attributes of the domain object in the visual representation of the surrogate. This way, surrogate interaction makes it possible to maintain a high degree of compatibility even for very rich domain object mappings.

The surrogate is a meta-object or archetype of its domain object. That means that it is not required to carry actual meaning (or it may carry aggregated meaning for several objects). Its visual representation can be stylized and decorated with annotations and explanations on how to interpret the data and how to interact with the surrogate and the domain objects.

Manipulating Multiple Objects

Surrogates can be linked to just a single domain object, or to several objects. Any changes to a surrogate will affect all of its linked domain objects. We have already seen that manipulating multiple objects using direct manipulation is complex due to the ambiguity and low predictability of operations performed on multiple operands. Surrogate interaction makes multiple object manipulation more visible and more predictable in two important ways:

- **Cardinality:** Each surrogate shows the number and type of the domain objects that will be affected by any operations performed on the surrogate.
- **Scope:** Surrogates communicate the scope of changes to an attribute by indicating the range of values for the underlying domain objects (see visual data aggregation below).

Furthermore, object manipulation can have several levels of *multiplicity*; the most common levels are the following:

- **Single object:** A single domain object, such as changes to a shape surrogate affecting a single shape on the canvas;
- **Set of objects:** A set of domain objects, such as formatting changes to a word surrogate affecting the selected region in a word processor; or

- **Class of objects:** All instances of a particular class of domain objects, such as changes to a bar chart surrogate affecting all bar charts in a scatterplot.

Depending on the type of the application, it may not make sense to support object manipulation for all levels. In the vector editor example, however, we would allow all three types of multiplicity levels depending on the selection: if a single object is selected, then changes will only affect that object, but if multiple objects are selected, then changes will affect them all. It would also make sense to be able to select the rectangle class, for instance, in order to be able to modify all rectangle objects on the canvas simultaneously.

Manipulating Intangible Properties

Unlike domain objects, surrogate objects do not necessarily have to portray actual visual entities. For direct manipulation, even abstract objects must have some visual form—or the designer must choose one—but the surrogate model relieves designers from this burden by allowing them to create only surrogates for such entities. Furthermore, this feature of the surrogate interaction model also promotes dividing different concerns of an application into separate surrogates.

For example, in the vector editor, we can create an entity representing the drawing canvas, where the user can change grid lines and snapping options through direct manipulation operations. Another example may be to expose the spacing between shapes as an actual tangible attribute in the shape surrogate—again, the user can directly change this spacing by manipulating the surrogate, something that would be difficult to achieve on the actual domain object.

Even properties that are difficult to manipulate using direct manipulation, such as choosing a color or selecting a stippling for a line, can be made more natural by situating the button to open the color selection dialog (or stippling dropdown box) on top of the the surrogate. While this does not improve the low directness of actually performing the task, it does increase the degree of compatibility of the interaction.

Visual Data Aggregation

Because the surrogate is an archetype of its linked domain objects (potentially several of them), it need not necessarily convey any specific information about the dataset. In our vector editor example, we may choose to show a stylized shape as a surrogate. On the other hand, this presents an opportunity in itself—the surrogate can be designed to serve as a visual representation of the aggregated data of its linked objects. In other words, for the vector editor example, we can use the surrogate shape to show data about the underlying shapes and their minimum and maximum line thickness, their different line and fill colors, and their relative sizes.

This concept can be extended to the domain object attributes controlled by user interface components in the surrogate. By using scented widgets similar to Willett et al. [36], we can communicate information about the number, range, and distribution of values for attributes controlled by a particular interface component. For example, a slider controlling line thickness in our vector drawing program may be combined with a histogram showing the totals of the linked shapes for each thickness value, exposing data about the drawing to the user. Of course, depending on the multiplicity level supported by the application, changing the value of such a widget would instantly change the underlying distribution of the linked objects (thereby instantly updating the histogram).

Cognitive Aspects

There is a fundamental cognitive difference between standard control panels and surrogate objects. Control panels are more or less haphazard collections of user interface controls (sliders, fields, buttons) and thus only have the most abstract relation to the domain object being manipulated. In contrast, a surrogate object—by virtue of the three characteristics above—embed the controls in interactions and visual representations that are *compatible* with the domain objects at a deep cognitive level. In other words, surrogate interaction allows us to strike a more precise and fine-grained balance between indirection and compatibility in the interface.

DESIGN EXAMPLES

In our experience, some general strategies are particularly useful when implementing actual surrogate interfaces:

- Surrogate objects can be seen as examples of both Prototype and Proxy design patterns [13]. They are prototypes because they act as archetypes of domain objects when instantiating new objects. They are proxies because all domain objects are manipulated through the surrogate.

- Domain objects can be seen as a simple registry of attributes (name-value pairs) controlled using surrogates. To support classes of domain objects, such as circles and squares in a drawing editor, objects can be arranged in a hierarchy that supports class narrowing of a set of objects.

In the examples below, we study surrogate objects applied to data visualization, vector drawing, and mobile interaction.

Scatterplot Visualization

Let us study how we can use surrogate interaction to augment a simple 2D scatterplot [6] visualization. A scatterplot is constructed by mapping two dimensions of the multidimensional dataset to visualize to the horizontal (X) and vertical (Y) axes. Entities in the dataset can now be drawn as points in the Cartesian space formed by the axes. Displaying additional dimensions can be done by varying the graphical properties of the points themselves, such as using their color, shape, and size to convey additional data. However, there is generally a limit to what these graphical properties can communicate. An alternative approach that may be more scalable is to turn the points themselves into miniature bar chart glyphs capable of displaying additional dimensions [1].

At this point, the scatterplot visualization is clearly a graphical application with rich domain objects: the axes and the bar charts. Controlling all of these attributes using direct manipulation on the visualization canvas is difficult due to the high data density and visual clutter. Instead, the practice in visualization applications is to provide menus and control panels where users can indirectly manipulate these settings.

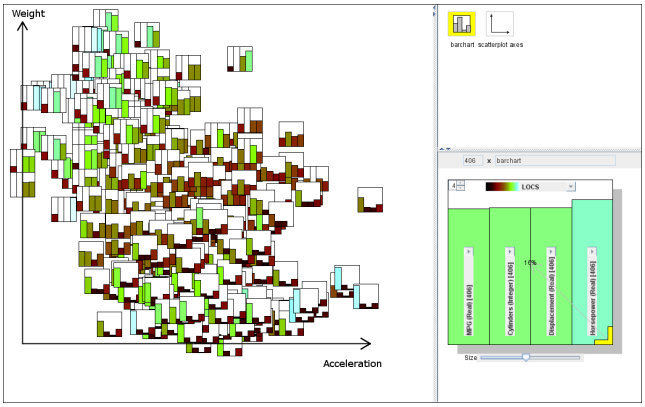

Figure 6. Scatterplot of the weight (Y) and acceleration (X) of around 400 cars. The bar charts show additional information about each car.

To improve this indirect mode of interaction, we have designed a scatterplot implementation that uses surrogate interaction to support access to all of the visualization attributes. Figure 6 shows a screenshot of the prototype. It exposes the two types of domain objects as surrogates in the palette:

- **Scatterplot axes:** The horizontal and vertical axes of the scatterplot, allowing the user to change which dimension to map to each axis as well as graphical attributes of the space (background color, tick marks, grid lines, etc).
- **Bar chart:** The bar charts that represent the points, allowing the user to control the number of bars (i.e., additional dimensions) to show for each data point, the color scale to use for the bars, and the size of the charts (Figure 7).

In this application, we need to maintain consistency across all bar charts, so we do not allow selecting individual glyphs. Instead, changing the surrogate will affect all bar charts.

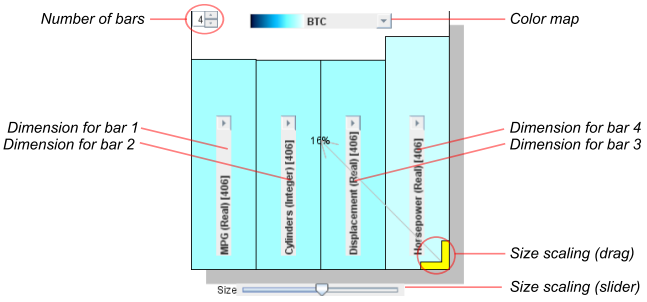

Figure 7. Bar chart surrogate for the scatterplot application.

Starplot Visualization

A starplot [29] is a multidimensional visualization where each item in a dataset becomes a circular glyph with its dimensions arranged around the radius of the glyph. A closed polygon connects values for each of the dimensions mapped to the radial axes. By creating a starplot for each data point in a dataset and laying them out on the visual space, the viewer can easily detect outliers and trends in the data.

Again, as is often the case in visualization, this is an example of an application consisting of heavyweight and attribute-rich domain objects. Instead of building menus or control panels to modify them, we apply surrogate interaction to create a visualization based on two surrogate objects (Figure 8):

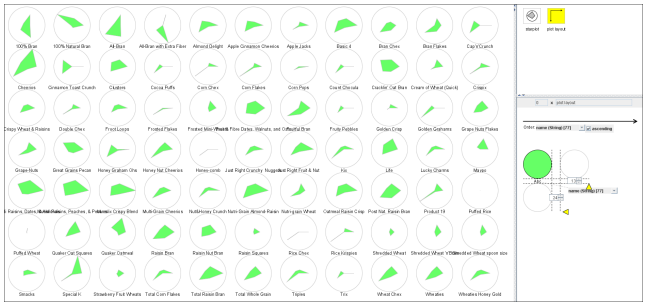

Figure 8. Starplots for 77 different cereals. Each plot shows calories, fiber, sodium, fat, and protein content for each cereal.

- **Starplot:** The starplot entities, enabling the user to change the number of dimensions to display and the mapping for each radial axis. Users can also use the surrogate to change color mapping and the glyph size.
- **Plot layout:** Layout is an intangible attribute that has no visual representation, but this surrogate enables users to change both horizontal and vertical glyph spacing as well as the sorting order for glyphs on the visualization canvas.

In particular, the layout surrogate (Figure 9) is useful because it provides a natural place for exposing intangible attributes. Glyph spacing could be seen as a glyph attribute (the amount of padding around each glyph), but the sorting order does not make sense anywhere else than in the layout surrogate (the standard solution is a menu or control panel).

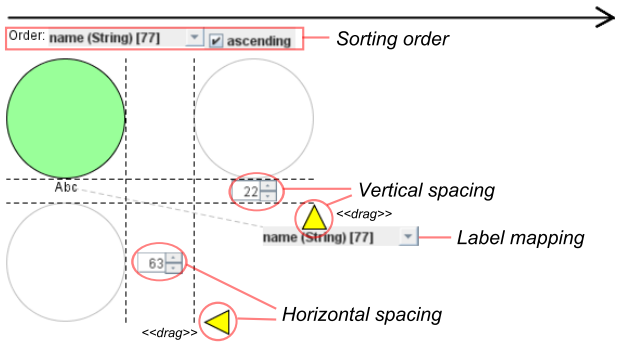

Figure 9. Plot surrogate object for the starplot application.

Vector Drawing Editor

Vector drawing editors have already been discussed extensively in this paper as an example of precisely the kind of application that is most suitable for surrogate interaction. Therefore, we have built such a vector editor based on surrogate interaction concepts. Figure 10 shows the basic structure the editor, closely mirroring that of the scatterplot and starplot applications. In this screenshot, the user has utilized the tools in the toolbar (top) to draw a set of shapes on the main canvas. The surrogate is showing that two shapes have been selected—note the visual aggregation in the surrogate

representation for color and line thickness—and is exposing controls to change both shapes collectively.

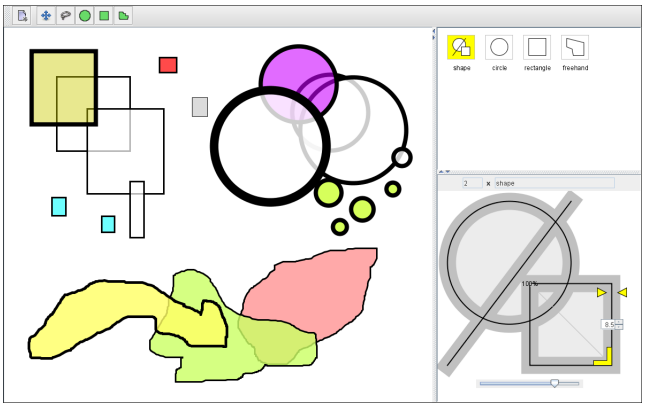

Figure 10. Vector drawing editor using surrogate interaction.

The surrogates that we support in the editor are organized in an inheritance hierarchy with the abstract **shape** class at the top—all concrete shapes inherit from this class. This makes for a structure that can be easily expanded for a realistic vector editor. When selecting shapes on the canvas using the lasso tool, the editor will automatically narrow the surrogate to the lowest common ancestor to the selected shapes in the shape hierarchy. This can clearly be seen in Figure 10—because the selected shapes are polymorphic, the surrogate has only been narrowed to the common shape base class.

The editor currently supports the following surrogate types:

- **Shape:** This surrogate exposes attributes that are common to shapes, such as line width, color, and transparency.
- **Circle:** Manipulation of the radius.
- **Rectangle:** Manipulation of height and width.
- **Freehand:** Editing individual vertices.

Figure 11 shows the surrogates in the vector editor. For some attributes, we expose both numeric and direct manipulation controls (signified by yellow shapes that the user can click and drag—e.g., Line thickness and Size scaling in the figure—whereas other attributes only have an indirect control. However, in this situation, the control is located in the context of its effect, thus minimizing indirection.

Interaction on Mobile Devices

Ubiquitous computing is on the rise, and concepts like *embodied interaction* [8] and *tangible computing* [14] are augmenting physical objects with computational abilities. However, the real world has the same limitations as the digital one: we still do not want to overload physical objects with too many controls, and we often lack metaphors for how to control many objects simultaneously. Surrogate interaction could be used to create surrogate interfaces of physical objects on mobile devices to control them (Figure 12).

IMPLICATIONS FOR DESIGN

There is no general panacea that resolves all issues of interaction design. Rather, interaction in an application should be

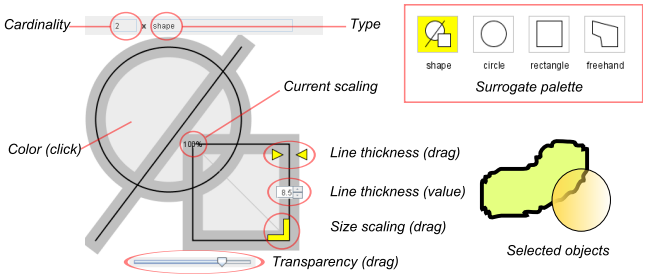

Figure 11. Basic components of the surrogate drawing editor. The surrogate palette (top right) is used to select different shape classes. The surrogate (left) is used to control the selected shapes (bottom right). Absolute and relative scaling can be toggled by clicking on the size indicator (percentage), giving a clear indication of scaling behavior.

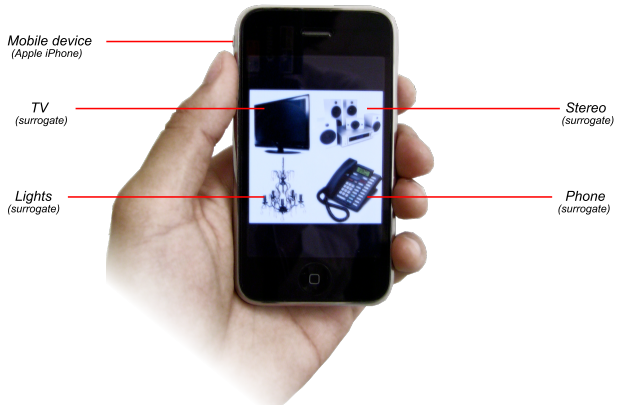

Figure 12. Surrogate interface on a smartphone for interacting with home electronic devices using surrogates (mockup).

analyzed so that the appropriate interaction model (or models) can be chosen. While our design examples demonstrate that surrogate interaction can benefit many applications, we should note that the method is a complement to and not a replacement for direct manipulation and other methods. The most beneficial use of surrogate interaction is often in combination with direct manipulation, delegating complex, group-based, or intangible manipulation to a surrogate.

Some tasks are simply not suitable for surrogate interaction due to the information loss implicit in creating a surrogate object. Below we outline some of the tradeoffs that must be considered when applying surrogate interaction in a design:

- **Increased indirection.** Surrogate interaction clearly increases the spatial indirection in the interface.
- **Divided attention.** Surrogates are separate from the main display, forcing users to split their attention between them.
- **Ex situ.** Surrogates cannot convey positions because they are removed from their context, so absolute operations (e.g., move and resize) are best left to direct manipulation.
- **Visual representation.** Choosing a good visual representation, especially for abstract entities, can be challenging.

We discuss some aspects of these limitations and how to address them in the following subsections.

Surrogate Interaction in Context

There is already some evidence of surrogate interaction in the instrumental interaction model. Beaudouin-Lafon [3] discusses *meta-instruments* that operate on or organize other instruments; examples include toolboxes, menus, and instruments that modify other instruments. Surrogate interaction can be seen as an instance of *meta-objects*, a similar lifting of domain objects to a higher abstraction level.

In other words, a surrogate object is a reification of the abstract nature of domain objects. This abstraction enables surrogate interaction to more efficiently overcome the limitations of direct manipulation than other methods by better maintaining the cognitive link between meta-object and domain object. This way, despite introducing spatial indirection, surrogate interaction does not necessarily sacrifice the directness of the interaction; in fact, we believe that it may even *improve* directness in certain situations.

Activating Surrogates

Our design so far has positioned the surrogate object to the side of the main display, thus giving rise to the divided attention and ex situ problems listed above. However, it is possible to modify this design to remedy some of these issues by making the surrogate object activation dynamic:

- **Graphical context menu.** Similar to current textual context menus, typically activated by right-clicking on a certain object or area of the screen, we design our surrogate objects to appear as graphical context menus on selected domain objects. The obvious drawback is naturally that the surrogate object is invisible at all other times.
- **Trailing widget.** The surrogate object could be designed as a semi-transparent trailing widget [10] that follows the cursor, similar to the mini toolbar in Microsoft Office 2007. The surrogate would provide in-situ visual feedback without being disruptive, and a quick gesture would allow the user to acquire the surrogate for interaction.
- **In-situ replacement.** A third solution at the boundary between surrogate interaction and standard direct manipulation is to integrate the surrogate object with the domain object in-situ, allowing for the domain object itself to be replaced by the surrogate through an explicit interaction. Similar to semantic zooming [19], this design would temporarily assign more display space to the surrogate when it is active. Note that many direct manipulation interfaces already use a variant of this idea, for example displaying handles on graphical objects only when they are selected.

Visual Representations

A general difficulty with direct manipulation is choosing visual representations [26, 28], and this is compounded for surrogate interaction where the surrogates may represent multiple, abstract, or even intangible objects. For example, in our vector editor, we designed the visual representation of the abstract shape surrogate as a combination of a circle, a line, and a rectangle (Figure 10). This may be even more difficult for intangible attributes, such as spacing, sorting, or layout. In such instances, we found ourselves using indirect widgets such as combo boxes or spinners to control such attributes.

However, by situating widgets in the context of their use—e.g., on the surrogates—we can still minimize the cognitive and spatial indirection. Moreover, surrogate interaction frees designers from having to create visual representations suitable for use in the actual display, a difficult task for abstract objects. Surrogates for abstract objects, on the other hand, can be heavily annotated to ease understanding and usage.

CONCLUSIONS AND FUTURE WORK

We have presented surrogate interaction, a framework that unifies a large set of existing commercial and academic techniques into a comprehensive interaction model. In the model, users interact with surrogates of an application's domain objects. Actions to a surrogate will affect the connected domain object. This avoids cluttering the display with the multitude of controls necessary to interact with attribute-rich objects. Furthermore, the surrogates can be used to interact with multiple objects simultaneously, and can even be created for intangible objects, such as layout or spacing, and for attributes that only exist in the context of a group, such as sorting order or alignment. We have also presented design examples that showcase the interaction model.

The design space defined by direct manipulation and instrumental interaction is clearly very large, and surrogate interaction is just one example of a part of this space that can be profitable for certain situations and for certain applications. We will continue to explore surrogate interaction and its applications in the future, but also the whole design space of this area. We also plan to investigate the concept of meta-instruments and their corresponding meta-objects.

REFERENCES

1. N. Andrienko and G. Andrienko. *Exploratory Analysis of Spatial and Temporal Data: A Systematic Approach*. Springer-Verlag, 2006.
2. Apple Inc. Keynote. `http://www.apple.com/iwork/keynote/`.
3. M. Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 446–453, 2000.
4. M. Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the ACM Conference on Advanced Visual Interfaces*, pages 15–22, 2004.
5. B. Buxton. HCI and the inadequacies of direct manipulation systems. *SIGCHI Bulletins*, 25(1):21–22, 1993.
6. W. S. Cleveland and M. E. McGill, editors. *Dynamic Graphics for Statistics*. Wadsworth & Brooks/Cole, Pacific Grove, CA, USA, 1988.
7. P. R. Cohen, M. Dalrymple, D. B. Moran, F. C. N. Pereira, J. W. Sullivan, R. A. G. Jr, J. L. Schlossberg, and S. W. Tyler. Synergistic use of direct manipulation and natural language. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 227–233, 1989.

8. P. Dourish. *Where the Action Is: The Foundations of Embodied Interaction*. MIT Press, 2001.

9. C. Esposito, W. B. Paley, and J. Ong. Of mice and monkeys: A specialized input device for virtual body animation. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*, pages 109–114, 1995.

10. C. Forlines, D. Vogel, and R. Balakrishnan. HybridPointing: fluid switching between absolute and relative pointing with a direct input device. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 211–220, 2006.

11. D. Frohlich. Direct manipulation and other lessons. In M. Helander, T. K. Landauer, and P. V. Prabhu, editors, *Handbook of Human-Computer Interaction*, pages 463–488. 1997.

12. D. M. Frohlich. The history and future of direct manipulation. *Behaviour & Information Technology*, 12(6):315–329, 1993.

13. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

14. H. Ishii and B. Ullmer. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 234–241, 1997.

15. R. J. K. Jacob, A. Girouard, L. M. Hirshfield, M. S. Horn, O. Shaer, E. T. Solovey, and J. Zigelbaum. Reality-based interaction: a framework for post-WIMP interfaces. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 201–210, 2008.

16. C. N. Klokmose and M. Beaudouin-Lafon. VIGO: instrumental interaction in multi-surface environments. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 869–878, 2009.

17. P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, July 1994.

18. D. A. Norman. *The Design of Everyday Things*. The MIT Press, 1988.

19. K. Perlin and D. Fox. Pad: An alternative approach to the computer interface. In *Computer Graphics (Proceedings SIGGRAPH'93)*, pages 57–64, 1993.

20. M. P. Peterson. Active legends for interactive cartographic animation. *International Journal of Geographical Information Science*, 13(4):375–383, 1999.

21. J. S. Pierce, B. C. Stearns, and R. Randy Pausch. Voodoo dolls: Seamless interaction at multiple scales in virtual environments. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*, pages 141–145, 1999.

22. I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa. The go-go interaction technique: Nonlinear mapping for direct manipulation in VR. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 79–80, 1996.

23. J. Rekimoto. Pick-and-drop: A direct manipulation technique for multiple computer environments. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 31–39, 1997.

24. N. H. Riche, B. Lee, and C. Plaisant. Understanding interactive legends: a comparative study with standard widgets. *Computer Graphics Forum*, 29(3):1193–1202, 2010.

25. B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour and Information Technology*, 1(3):237–256, 1982.

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

27. B. Shneiderman and P. Maes. Direct manipulation vs. interface agents. *Interactions*, 4(6):42–61, 1997.

28. B. Shneiderman and C. Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 5th edition, 2009.

29. J. H. Siegel, E. J. Farrell, R. M. Goldwyn, and H. P. Friedman. The surgical implication of physiologic patterns in myocardial infarction shock. *Surgery*, 72(1):126–141, 1972.

30. R. Stoakley, M. J. Conway, and R. Pausch. Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 265–272, 1995.

31. The Omni Group. OmniGraffle Professional. `http://www.omnigroup.com/products/omnigraffle/`.

32. M. E. Tudoreanu and D. Hart. Interactive legends. In *Proceedings of the ACM Southeast Regional Conference*, pages 448–453, 2004.

33. E. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.

34. E. Tufte. *Beautiful Evidence*. Graphics Press, 2006.

35. A. van Dam. Post-WIMP user interfaces. *Communications of the ACM*, 40(2):63–67, Feb. 1997.

36. W. Willett, J. Heer, and M. Agrawala. Scented widgets: Improving navigation cues with embedded visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1129–1136, Nov./Dec. 2007.

37. C. Williamson and B. Shneiderman. The Dynamic HomeFinder: Evaluating dynamic queries in a real-estate information exploration system. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval*, pages 338–346, 1992.

38. Yahoo! Inc. Yahoo pipes. `http://pipes.yahoo.com`.