

A Semantic Analysis of the Design Space of Input Devices

**Jock Mackinlay, Stuart K. Card, and
George G. Robertson**
Xerox Palo Alto Research Center

ABSTRACT

A bewildering variety of devices for communication from humans to computers now exists on the market. In this article, we propose a descriptive framework for analyzing the design space of these input devices. We begin with Buxton's (1983) idea that input devices are transducers of physical properties in one, two, or three dimensions. Following Mackinlay's semantic analysis of the design space for graphical presentations, we extend this idea to more comprehensive descriptions of physical properties, space, and transducer mappings. In our reformulation, input devices are transducers of any combination of linear and rotary, absolute and relative, position and force, in any of the six spatial degrees of freedom. Simple input devices are described in terms of semantic mappings from the transducers of physical properties into the parameters of the applications. One of these mappings, the resolution function, allows us to describe the range of possibilities from continuous devices to discrete devices, including possibilities in between. Complex input controls are described in terms of hierarchical families of generic devices and in terms of composition operators on simpler devices. The description that emerges is used to produce a new taxonomy of input devices. The taxonomy is compared with previous taxonomies of Foley, Wallace, and Chan (1984) and of Buxton (1983) by reclassifying the devices previously analyzed by these authors. The descriptive techniques are further applied to the design of complex mouse-based virtual input controls for simulated three-dimensional

Authors' present address: Jock Mackinlay, Stuart K. Card, and George G. Robertson,
Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304.

CONTENTS

1. INTRODUCTION
 2. INPUT DEVICES
 - 2.1. A Simple Device
 - 2.2. Primitive Input Devices
 - 2.3. Connections
 - 2.4. Generic Devices
 - 2.5. Composition Operators for Input Devices
 - Examples of Composed Input Devices
 - 2.6. A Taxonomy of Input Devices
 - 2.7. Evaluating the Expressiveness of an Application's Input Devices
 3. SIMULATED 3D EGOCENTRIC MOTION CONTROLS: ANALYSIS AND GENERATION
 - 3.1. Egocentric Motion
 - 3.2. Analysis of an Existing Egocentric Motion Design
 - 3.3. Generation of Novel 3D Movement Controls: 3D Rooms
 - Step 1: Application Functionality
 - Step 2: Choosing Input Devices
 - Step 3: Designing Composite Input Controls
 4. DISCUSSION
 5. CONCLUSION
 - APPENDIX. DETAILED DESCRIPTION OF INSECT CONTROLS
 - Application Parameters
 - Basic Input Devices
 - Connections
 - Composite Insect Device
-

(3D) egocentric motion. One result is the design of a new virtual egocentric motion control.

1. INTRODUCTION

Over the last 20 years, the technology of human-computer interface has been evolving from a loose collection of techniques toward an engineering discipline. Advances in interface technology have led to a remarkable improvement in the quality of human-computer interfaces, although, of course, important shortcomings remain. The development of interface techniques is now at a point where it is appropriate to systematize existing research results and craft into a body of engineering and design knowledge. This is just part of the normal early development of a new engineering area.

Input devices are a case in point. A bewildering variety of devices for communication from humans to computers now exists on the market (see, e.g., Sherr, 1988). The range of input devices includes typewriter keyboards, mice, pens, tablets, "headmice," dialboxes, Polhemus cubes, gloves, and body

suits. Making sense of this hodgepodge of devices and systematizing the knowledge about them is certainly a challenge.

A popular approach to systematizing knowledge about input devices is the organization of human-computer dialogue techniques into collections of prebuilt interface modules called *user interface toolkits* or *user interface management systems*. User interface toolkits help with a wide range of problems, including the construction, runtime execution, and postruntime analysis of a user interface (Tanner & Buxton, 1985). For example, a toolkit may provide a library of prebuilt interface modules to control physical input devices (Olsen, Kasik, Rhyne, & Thomas, 1987; Pfaff, 1985), architecture and specification techniques for combining these modules (Anson, 1982; van den Bos, 1988), or postprocessing analysis tools (Olsen & Halversen, 1988).

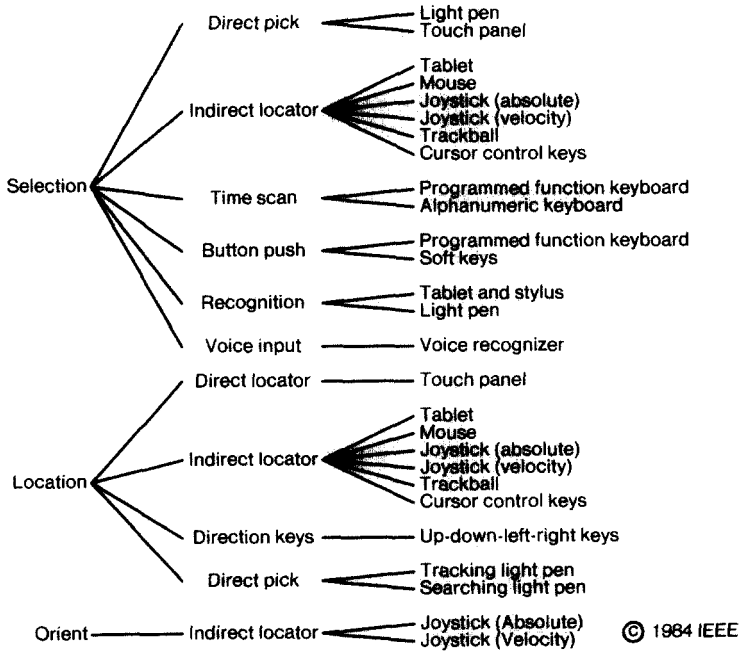
User interface toolkits help in the construction of interfaces and sometimes, as in Anson (1982), they provide architectural models of input device interactions. But the device models implicit in them sketch only a limited picture of the design space of input devices and their properties. Even for the construction of interfaces, they present interface designers with many design alternatives but do little to help with the design decisions themselves. The interface designer must still choose the appropriate modules from the toolkit library and decide how to combine them effectively (McCormack & Asente, 1988).

These design issues must also be addressed by the developer of a user interface toolkit, who must also deal with the size and complexity of the full design space of input devices. What should the toolkit library contain? How are interface modules going to be combined? Will it be easy to incorporate other physical or logical devices, such as the devices used to control 3D graphical applications? And, of course, what about input controls to nonworkstation devices such as radios, VCRs, or automobiles?

To achieve a systematic framework for input devices, toolkits need to be supported by technical abstractions about the nature of the task an input device is performing (Mackinlay, 1988; Newell & Card, 1986). In particular, these abstractions should help with both the generation of alternative designs and the evaluation of these alternatives. Of initial concern should be the generation of alternatives from the design space of input devices. Design alternatives cannot be evaluated until they are generated.

Two recent attempts have been made to provide abstractions that systematize the design space of input devices. Foley et al. (1984) focused on computer graphics subtasks. They developed trees like those shown in Figure 1 to describe the relationship between computer graphics tasks at the roots of the trees and input devices at the leaves. They also reviewed experimental evaluations of input devices. Buxton (Baecker & Buxton, 1987; Buxton, 1983) has shown that most input devices can be classified according to the physical properties and the number of spatial dimensions they sense, as in Figure 2. We can summarize Buxton's insight as:

Figure 1. Trees, adapted from Foley, Wallace, and Chan (1984), show the relationship between some computer graphic input applications and various input devices. From "The Human Factors of Computer Graphics Interaction Techniques" by J. D. Foley, V. L. Wallace, and P. Chan, 1984, *IEEE Computer Graphics & Applications*, 4(11), pp. 21-22. Copyright 1984 by IEEE. Adapted by permission.



Basically, an input device contains a transducer for an elementary physical property it can sense in one, two, or three spatial dimensions.

The elementary physical properties Buxton identified are position, motion, and pressure.

Both the Buxton (1983) and the Foley et al. (1984) taxonomies allow us to relate individual input devices to the space of possible designs. Foley, Wallace, and Chan's trees are the cross product of graphics tasks with input devices. This relates devices to tasks, but because single devices appear many times in the leaves of their trees, it is not easy to understand the similarities among devices. Their purpose was to give some initial organization to the major input devices that exist, rather than to claim that their collection of input devices is complete or systematically generated. Buxton's taxonomy takes a major step forward. By analyzing input devices as the transduction of


















Figure 2. Table for input device classification adapted from Buxton (1983), which is based on the property sensed, the number of dimensions sensed, and the sensing type, which distinguishes between Type T for devices that work by touch and Type M for devices that require a mechanical intermediary. From "Lexical and Pragmatic Considerations of Input Structures" by W. Buxton, 1983, *Computer Graphics*, 17(1), p. 34. Copyright 1983 by W. Buxton. Adapted by permission.

		Number of Dimensions							
		1		2		3			
Property sensed	Position	Rotary Pot	Sliding Pot	Tablet	Light Pen	Joystick	3D Joystick	M	Sensing type
		-----		Touch Tablet	Touch Screen	-----		T	
	Motion	Continuous Rotary Pod	Treadmill Thumbwheel	-----		Trackball	Trackball	M	
		-----		-----		Tasa X-Y Pad	-----	T	
	Pressure	Torque Sensing	Pressure Pad	-----		Isometric Joystick	-----	T	
		-----		-----		-----		-----	

basic physical properties, he is able to organize part of the basic design space. Although his coverage is systematic, it is limited to continuous physical devices. Neither taxonomy deals with the combination of individual input devices into complex input controls.

This article extends Buxton's analysis using an approach developed by Mackinlay (1986a, 1986b) to describe the design space for graphical presentations of relational data (e.g., bar charts, scatter plots, and connected graphs). Mackinlay treated graphical presentations as sentences of artificial languages. A semantic analysis of these graphical languages was used to develop two combinatoric techniques (see Figure 3) for describing a wide variety of alternative designs. The first technique was to develop a set of primitive graphical languages based on orthogonal graphical properties, such as position, color, and shape, derived from the cartographer Bertin (1983). These graphical languages act as generic building blocks for describing the design space. The second technique was to develop composition operators for combining these building blocks to convey a rich set of meanings. Using this approach, we reformulate and extend Buxton's properties to propose a new basis set of physical properties that underly most input devices. We then follow mappings of information from the raw physical transducers of an input

Figure 3. Mackinlay (1986a, 1986b) derived a small set of primitive graphical techniques from the cartographer Bertin (1983) and developed three composition operators that support the automatic generation of a wide range of graphical presentations.

Primitive techniques	Composition operators	Composite design
Marks Points Lines Areas   	Single Axis   	
Position 	Double Axis   	
Retinal Size 	Mark   	

device through the device and into the semantics of an application. Following Mackinlay, these mappings can be evaluated in terms of two types of evaluation criteria for comparing alternative designs: *expressiveness* (the sentence conveys exactly the intended meaning) and *effectiveness* (the sentence conveys the intended meaning with felicity). In this article, we evaluate the mappings for expressiveness. The result is a semantic task analysis that gives a new classification for a wide variety of input controls, including the previous taxonomies of Buxton (1983), Foley et al. (1984), and some mouse-based input controls for simulated 3D egocentric motion. The evaluation of effectiveness is left for future work (Card, Mackinlay, & Robertson, 1990).

2. INPUT DEVICES

An input device is part of the means used to engage in dialogue with a computer or other machine. The dialogue is not, of course, in natural language but is conducted in ways peculiarly suited to interaction between human and machine. Unlike human-human conversation, the dialogue is between fundamentally dissimilar agents—in terms of both perception and processing. The human makes utterances to the machine by twisting dials, pressing buttons, moving a mouse, or performing other manipulations of the physical world. These are signals easily communicated to the circuitry of the machine. The machine may reply in displays or sounds easily communicated to the perceptual apparatus of the human. The dialogue may be organized to simulate interaction through an “agent,” or it may be organized to simulate the use of a “tool.” The principle job of the designer of such machines is the design of artificial languages to communicate meaning between the human

and the machine. Twisting a dial or pressing a sequence of buttons represents sentences in these languages.

Following Mackinlay (1986a, 1986b), we assume that the designer develops languages so as to fulfill expressiveness and effectiveness criteria. In general, understanding these two criteria requires different supporting sciences:

1. To analyze expressiveness, we use results from formal language theory in computer science. In particular, we say:

A human-machine artificial language can be used to express some information when it can encode exactly the input information, that is, all the information and only the information. (Mackinlay, 1986a, 1986b)

One of the most difficult parts of artificial language design is to prevent the language from expressing additional, unintended information (Mackinlay & Genesereth, 1985).

2. To analyze effectiveness, we use results from the human sciences. Various approaches are possible. For example, we could use psychology for expected errors or speed in receiving the information or for difficulty of learning the language. We could use cognitive models if available or just data if necessary and generalizable enough. We could also use results from the social sciences. Or, we could use other criteria, such as cost or panel space. Finally, the pragmatics of the application itself might be used to weigh the importance of different effectiveness criteria. Although effectiveness of input devices is beyond the scope of this article (however, see Card, Mackinlay, & Robertson, 1990), there are clearly defined complementary roles for both machine and human sciences in the design of artificial languages.

In this article, we concentrate on the expressiveness of an input device—on the ways in which physical manipulation can communicate meanings (in this case, parameter values) to an application. The resulting semantic analysis of the flow of information from a user's manipulations through an input device to an application reveals the basic structure of the design space of input devices. We start with a simple set of controls that encapsulate the important issues.

2.1. A Simple Device

Figure 4 contains a picture of a radio with three knobs for controlling its volume, selecting power off and frequency band, and tuning in a station. The easiest of these to describe is the volume knob. Physically, the user manipu-

Figure 1. Tabular summary of the contributions of each of the four broad research initiatives described in this article as seen from five differing viewpoints on the key issues in human-computer interaction.

Viewpoint	Initiative			
	Understanding Goals and Preferences	Broadening Applied Cognitive Theory	Supporting Innovation	Credit Assignment
Cumulative psychological science	Prerequisite	Fills out vision	Assists application	Resource allocation
Not <i>that</i> kind	Prerequisite	Clarifies intuitions	Assists application	Resource allocation
Context	Clarifies central claim	Produces relevant theory	Assists application	Resource allocation
Process	Supports evaluation	Strengthens covert theory	Speeds up process	Resource allocation
Systems	Supports generalization	Strengthens covert theory	Feeds analysis	Resource allocation

Now consider the selection knob in Figure 4, which selects *OFF*, *AM*, or *FM*. This selection knob also rotates about the *Z* axis, but it senses this rotation with a discrete resolution. To model this, we use a *resolution function* that maps the continuous domain of the control into only certain values (refinements are possible to model controls that spring to different positions):

SelectorKnob is defined as
 a rotation about the *Z* axis
 between 0° and 90°
 with a resolution that maps the continuous region
 into the discrete set $\{0^\circ, 45^\circ, 90^\circ\}$.

This device is connected to a discrete application parameter:

The connection from the **SelectorKnob** to the radio **AMTuner** and **FMTuner**
 maps from a rotation of $\{0^\circ, 45^\circ, 90^\circ\}$
 to the set $\{OFF, ON\}$ for both tuners.

Finally, consider the more complicated station knob. It works through a string mechanism that connects it to the slider shown in Figure 4, which in turn is connected to the tuners. The result is a knob that continues to rotate even when the slider has been moved to its maximum or minimum position; yet when the direction of knob rotation is reversed, the position of the slider immediately begins to change. Here it is not the absolute angle of the knob that is important but the relative change of the angle during manipulation:

StationKnob is defined as
 a relative rotation about the *Z* axis
 between $-\infty$ and $+\infty$ degrees
 with a continuous resolution.

Instead of being mapped onto an application directly, the station knob is cascaded onto another input device, the station slider:

The connection from the **StationKnob** to the **StationSlider**
 maps from change in rotation in degrees
 to position in inches.
StationSlider is defined as
 a positioning on the *X* axis
 between 0 in. and 5 in.
 with a continuous resolution.

Figure 5. Physical properties sensed by input devices.

	Linear	Rotary
Position		
Absolute	Position P	Rotation R
Relative	Movement dP	DeltaRotation dR
Force		
Absolute	Force F	Torque T
Relative	DeltaForce dF	DeltaTorque dT

The station slider, in turn, is connected to the radio application:

The connection from the **StationSlider** to the radio **AMTuner** and **FMTuner**

maps from position in inches

to frequency in Hz above some base frequency.

Notice that the absolute angle of the **StationKnob** itself is not of significance, unlike the case of the **VolumeKnob**, because of the play between the **StationKnob** and the **StationSlider** at the **StationSlider's** endpoints. Instead, it is the change in the angle that is important.

2.2. Primitive Input Devices

Our simple radio example informally illustrates some of the important parts of an input device: the geometry of the transducers of physical manipulation, the domain of values that the transducer can produce, device resolution, and connections among devices. Now we are ready to make these distinctions more precise, to which purpose we propose a theory that formally describes a wide range of input devices. The theory focuses on the semantic information that must be communicated by input devices from a user to the application. We start with Buxton's taxonomy described in Figure 2. The virtue of Buxton's scheme is that it relates input devices back to the elementary physical and spatial properties these devices transduce. This allows us to trace a manipulation language from its roots in the elementary properties of physical objects and space, to manipulations of those physical properties, to the delivery of meaningful sentences of information to an application. We extend Buxton's scheme to further abstract the set of physical and spatial properties and to be able to describe discrete, composite, and virtual devices. Our set of elementary physical properties includes all possible combinations of linear and rotary as well as absolute and relative values of spatial position and force, as shown in Figure 5. Instead of just counting the

number of dimensions involved as Buxton did, we distinguish the particular dimensions in a six-dimensional, user-based coordinate system: three linear and three rotational dimensions. Thus, the **VolumeKnob** on our radio example senses absolute rotation about the Z axis. A mouse, on the other hand, senses relative position in X and Y .

Abstractly, an input device is sensitive to user manipulation of physical properties in an input range, and it transduces a given input value into an output range. We define an *input device* to be a 6-tuple:

$$\langle M, In, S, R, Out, W \rangle.$$

The components of the “sixtuple” are defined as follows:

1. **M** is a *manipulation operator*. A manipulation operator corresponds to the physical property vector in Figure 5 that the device senses. We name manipulation operators by the corresponding physical property and use subscripts to describe the components of the sensed property. For example, R_z represents a rotation manipulation around the Z axis.

2. **In** is the *input domain set* over which a manipulation operator will sense a value. For example, the input domain for the **VolumeKnob** previously described is angles of the knob between 0° and 270° .

3. **S** is the current *state* of the device, including the external state of its input and output and the internal state used by the device.

4. **R** is a *resolution function* that maps from the input domain set to the output domain set. This allows us to model detents and variable device resolution. For example, the **SelectorKnob** previously described has a resolution function that maps from the set of real numbers representing angles of the knob to the set $\{0^\circ, 45^\circ, 90^\circ\}$. The resolution function can be the identify function I , in which case the values in the input domain set **In** are mapped directly to the output domain set **Out**. (This happened for the **VolumeKnob** previously described.)

5. **Out** is the *output domain set*, which describes the range of the resolution function, the set of values into which the input domain set is mapped by the device.

6. **W** is a general purpose set of device properties that describes additional aspects of how the input device *works*, such as its physical characteristics or its internal mechanism. In particular, **W** contains a list of production rules that trigger on internal and external states of the device and allow us to describe common properties of input devices, such as the detents of a knob, or spring-loaded push buttons. For example, a production can describe the spring of a conventional “return-to-zero” joystick. If we assume that the device state includes two Booleans, *Grasp* and *Release*, that indicate when the user starts and stops manipulating the device, the following rule describes the

effect of the spring, where In represents the instantaneous input state of the joystick:

$$\text{If } (Release = True) \text{ then } In := 0$$

That is, whenever the user releases the device, its input state returns to 0.

Another use for the productions is to describe an input device that interprets a manipulation in a relative manner. Abstractly, a relative input device is calculating $Out := In_{t_i} - In_{t_i - 1}$, that is, the difference of the value In of the input state of the device at two successive times $t_i - 1$ and t_i . However, the value of a past state is not available unless it is stored in an internal state variable. We can use a production to describe the mechanism that updates this internal variable.

The following is a formal description of the volume knob defined informally in Equation 1 (tuple arguments are labeled for ease of understanding):

$$\begin{aligned} \text{VolumeKnob} = & \\ & \langle \text{Manipulation} \quad R_z, \\ & \text{InputDomain:} \quad [0^\circ, 270^\circ], \\ & \text{State:} \quad \theta, \\ & \text{ResolutionFn:} \quad I, \\ & \text{OutputDomain:} \quad [0^\circ, 270^\circ], \\ & \text{Works:} \quad \{ \} >, \end{aligned} \quad (3)$$

where I is the identity function. As a shorthand notation, we typically write Equation 3 compactly as:

$$\begin{aligned} \text{VolumeKnob} = & \\ & R_z: [0^\circ, 270^\circ] - I \rightarrow [0^\circ, 270^\circ]. \end{aligned}$$

This shorthand notation reflects the fact that input devices are essentially functions from input domains to output domains. The notation also allows the inclusion of additional information about the resolution function, as in the following description of the radio station knob:

$$\begin{aligned} \text{StationKnob} = & \\ & R_z: [0^\circ, 90^\circ] - f \rightarrow \langle 0^\circ, 45^\circ, 90^\circ \rangle \\ & \text{where } f(In) = \\ & \quad [0^\circ, 22.5^\circ] \rightarrow \langle 0^\circ \rangle \\ & \quad [22.5^\circ, 67.5^\circ] \rightarrow \langle 45^\circ \rangle \\ & \quad [67.5^\circ, 90^\circ] \rightarrow \langle 90^\circ \rangle. \end{aligned}$$

We write *In* with italics to refer to the value of the input device at a particular instant in time—its input state. Helios is used for device names and sets. Thus, *In* refers to the set of values the input device can take—its input domain set.

2.3. Connections

Now that we have a notation for describing devices, the next step is to describe the connections between devices and application parameters. Because we are interested in describing virtual and composite devices, as well as physical devices, we treat connections (and, in fact, applications such as the radio) as just other specialized devices and use the same formal machinery. In these cases, we no longer require that the manipulation operators represent manipulations of physical properties. We use the term *device* in the general case and reserve the term *input device* for cases where the operators do represent manipulations of physical properties (which includes virtual input devices).

Although it is possible to describe connections and application parameters with full sixtuples, it is sometimes convenient to omit details irrelevant to the discussion. For the application devices, the input domain is generally sufficient for our needs. So we use:

Volume =
 <InputDomain: [0, 25] decibels> ,

to refer to the loudness of sound the radio actually makes. As a notational convenience, we often use the name of the device to refer to this domain set, in this case **Volume**.

For connection devices, which map the output domain set of one device to the input domain set of the other device, we only need three parameters that specify the output domain of the first device, the mapping function, and the input domain of the second device. For example, the connection between the **VolumeKnob** input device and the radio **Volume** application parameter, given informally in Equation 2, can be formally described as:

Connect =
 <InputDomain: [0°, 270°],
 ResolutionFn: f ,
 OutputDomain: [0, 25] decibels> .

As a notational convenience, we avoid the repetition of domain set descriptions in the input devices and the connection and use the following ternary

predicate, where the name of the device is used to refer to the corresponding domain set, so the aforementioned expression could be written as:

$$\text{Connect}(\text{VolumeKnob}, \text{Volume}, \\ f(\theta \text{ degrees}) = C_v \times \theta \text{ decibels}).$$

This expression can be read "the connection device maps from the output domain set of **VolumeKnob** to the input domain set of **Volume** using the function $f(\theta) = C_v \times \theta$," where C_v is a constant of proportionality determined by the gain of the control and conversion factors among the units of measurement.

Given the following application parameters for the radio:

Volume:	[0, 25] decibels
AMTuner:	[530, 1610] kHz X {OFF, ON}
FMTuner:	[88.1, 107.3] MHz X {OFF, ON},

we can formally describe the radio connections as follows:

RADIO DEVICE CONNECTIONS:

Connect (**VolumeKnob**, **Volume**,
 $f(\theta \text{ degrees}) = C_v \times \theta \text{ decibels}$),

Connect (**SelectionKnob**, **AMTuner**,
 $f(\theta \text{ degrees}) =$
 $\{45^\circ\} \rightarrow \text{ON}$
 $\{0^\circ, 90^\circ\} \rightarrow \text{OFF}$),

Connect (**SelectionKnob**, **FMTuner**,
 $f(\theta \text{ degrees}) =$
 $\{90^\circ\} \rightarrow \text{ON}$
 $\{0^\circ, 45^\circ\} \rightarrow \text{OFF}$),

Connect (**StationKnob**, **StationSlider**,
 $f(\theta \text{ degrees}) = C_t \times \theta \text{ inches}$)

Connect (**StationSlider**, **AMTuner**,
 $f(x \text{ inches}) = C_a \times x \text{ Hz}$)

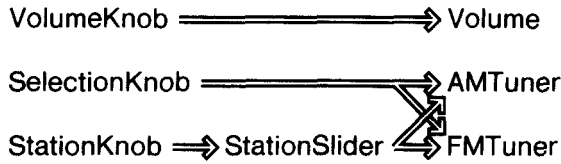
Connect (**StationSlider**, **FMTuner**,
 $f(x \text{ inches}) = C_f \times x \text{ MHz}$).

(4)

The connections are shown diagrammatically in Figure 6.

Note that although our radio example has scaling occurring in the connections, it is possible for devices to do scaling in the device itself or even in the application. All three cases can be modeled in this notation by moving the mapping function to the appropriate location.

Figure 6. The devices and connections for the simple radio.



2.4. Generic Devices

We use the idea of generic input devices to describe families of similar devices in the design space of input devices. The radio knobs in our example, for instance, are all similar. They represent specializations of a prototypical knob,

$$\text{GenericKnob}_z = R_z: \text{In} -I \rightarrow \text{Out}, \quad (5)$$

which senses rotation around a Z axis—different specializations of this generic knob lead to different knobs. The device in Equation 5 is a *generic device*, a device sextuple where some of the arguments have not been completely specified. We define specific devices as instantiations of the generic device, using the predicate:

$$\text{Instantiate}(\text{GenericDevice}, \text{DeviceArguments}).$$

Partially specified tuple arguments lead to additional arguments that must be specified when the generic device is instantiated. The extended notation thus has an object-oriented character. The In and Out sets in the generic knob in Equation 5 could be partially specified with additional arguments Min_z and Max_z to describe a generic knob that has bounds on its input values:

$$\text{GenericBoundedKnob}_z = R_z: [\text{Min}_z, \text{Max}_z] -f \rightarrow [f(\text{Min}_z), f(\text{Max}_z)].$$

The **VolumeKnob** previously specified in Equation 3 can be described as an instantiation of this generic bounded knob:

$$\begin{aligned} \text{VolumeKnob} = \\ &\text{Instantiate}(\text{GenericBoundedKnob}_z, \\ &\quad \text{In: } [0^\circ, 270^\circ], \text{R: } I). \end{aligned}$$

The idea of generic devices is a useful design aid for exploring the design space of input devices before having to contend with the details of specific

input devices. For example, knobs can also rotate freely, sample from a fixed range, or only take on discrete input values:

GenericFreeKnob_z =
 $R_z: \text{Real} \rightarrow \text{Real},$
GenericSamplingKnob_z =
 $R_z: [Min_z, Max_z]$
 $\rightarrow s \mapsto \langle s(Min_z), \dots, s(Max_z) \rangle$
GenericDiscreteKnob_z =
 $R_z: \langle Min_z, \dots, Max_z \rangle$
 $\rightarrow f \mapsto \langle f(Min_z), \dots, f(Max_z) \rangle.$

Note that physical knobs do not actually have a discrete input domain set because they must always be manipulated through a continuous range. However, physical knobs often include springs that force the knob to the closest discrete detent, giving the user the feedback of a discrete knob. (Formally, this is modeled with a production that changes the input state when the user stops manipulating the knob.) We use the tuple notation for a device's input domain set as a shorthand to indicate the existence of such feedback.

The only other generic device needed to describe the radio input devices is the following generic slider:

GenericSlider_x = $P_x: [Min_x, Max_x] \rightarrow [f(Min_x), f(Max_x)].$

Sampling and discrete generic sliders can be specified in the obvious way.

We can now redescribe the devices that appear in our venerable radio example as follows:

RADIO DEVICES:

VolumeKnob =
Instantiate (**GenericKnob_z**, In: $[0^\circ, 270^\circ]$, R: I)
SelectionKnob =
Instantiate (**GenericDiscreteKnob_z**, In: $[0^\circ, 90^\circ]$,
 Out: $\langle 0^\circ, 45^\circ, 90^\circ \rangle$)
StationKnob =
Instantiate (**GenericFreeKnob_z**)
StationSlider =
Instantiate (**GenericSlider_x**, In: $[0, 5]$),

where some device arguments have been omitted. For example, the Out set of **VolumeKnob** is clearly $[0^\circ, 270^\circ]$.

Before we turn to the description of composite input devices, including

two-dimensional devices, we describe the following two one-dimensional generic devices:

$$\begin{aligned}
 \text{GenericButton}_z &= \\
 P_z: [0, Max_z] &-f \rightarrow \langle Up, Down \rangle \\
 \text{Generic1D-Tablet}_x &= \\
 P_x: [Min_x, Max_x] &-f \rightarrow [f(Min_x), f(Max_x)] \cup \\
 RestState &-f \rightarrow Null.
 \end{aligned}$$

Tablets are sensitive over their entire input range to the position of a touch. Some tablets are sensitive only to the touch of specialized pens.

One difference between a tablet and a slider is that the tablet has a special input *RestState* when the user is not manipulating the device, and it reports a *Null* value when the device is at rest. By contrast, sliders maintain their input state when the user stops manipulating the slider.

In fact, tablets raise some interesting issues that have not yet been covered by our formal analysis. In particular, the state of a tablet can change without having to pass through intermediate values between the previous state and the new state, as was the case for the knobs we discussed—That is, the tablet is sensitive over its entire input range, not just in the neighborhood of the current state. This property is different from the one Buxton (1983) used in his taxonomy, where he distinguished between devices that require a mechanical intermediary instead of touch. A light pen requires a mechanical intermediary and is still sensitive over its entire input range. Three issues are raised by these considerations:

1. *Component acquisition.* Some devices require additional movement to a specific location to acquire a component of the device that is required for manipulation (e.g., a light pen, a mouse, or a slider knob). The need for additional movement is important for evaluating the effectiveness of input devices.

2. *Persistent state feedback.* Some devices maintain and feed back their state even when they are not being manipulated. For example, a slider knob indicates the slider's current state. Not all hand-movement devices, however, have this property. A mouse, for example, is sensitive to relative position, but its location at rest may not reveal any interesting state. Also, when a knob can be rotated through more than 360° , its current position does not reveal its state unless there is additional feedback. State feedback is an important effectiveness property.

3. *Sensitivity over input domain.* Some devices are sensitive over their entire input domain; that is, the user can specify a new disjoint value without passing through the set of values on the way to the new value (as would

happen with a radio volume control). This property is distinct from the other two.

These distinctions can be captured in our formalism using the works W argument of the device tuple.

2.5. Composition Operators for Input Devices

The discussion so far has focused on simple radio controls and various one-dimensional devices. But simple input devices can also be composed into a bewildering variety of high-dimensional devices and elaborate input controls. Given our semantic analysis of input devices, we have identified the following three composition operators for generating controls in the design space of input devices: (a) Two devices can be connected, as we have seen, so that the output of one is cascaded to the input of the other. The station knob and station slider compositions are examples of connection composition; (b) several independent devices can be laid out together in a control panel. For example, the radio has a panel of three knobs. The best known example of this sort of device is a typewriter, which comprises a set of more or less identical button devices laid out together, each associated with a different letter; or (c) two devices can be composed so that their domain sets are merged. For example, the position sensor of the mouse is a merged composition of two relative position sensors combined orthogonally. Our formalism allows us to describe each of these composition operators more precisely:

1. *Connection composition.* Devices are connected by cascading the output of one device to the input of the other. In this sort of composition operator, the output domain set from one device is mapped via a connection to the input domain set of another device. An example is the station knob cascade described previously in Equation 4,

$$\text{Connect}(\text{StationKnob}, \text{StationSlider}, \\ f(\theta \text{ degrees}) = C_i \times \theta \text{ inches}).$$

For convenience, we shorten this still further to:

$$\text{StationKnob} \Rightarrow \text{StationSlider}.$$

2. *Layout composition.* The layout of devices on a panel can be described with a mapping of the local coordinates of a device to the coordinates of the panel, usually involving translations and rotations. This mapping can be described as one of the physical properties of the composite device. Formally we use the

symbol \otimes to indicate a layout composition. For example, the layout of the left, middle, and right buttons to the top of a mouse can be described as:

$$\begin{aligned} & \text{LMouseButton} T_1(z) \otimes \\ & \text{MMouseButton} T_2(z) \otimes \\ & \text{RMouseButton} T_3(z), \end{aligned}$$

where the transformation functions T_1 , T_2 , and T_3 transform the coordinates of the respective button devices to the top of the mouse. (See the Appendix for details about the instantiation of these buttons from the **GenericButton_x** device.)

3. *Merge composition.* The final form of device composition is a special case of layout composition where the two devices are merged such that their combined domain sets are treated as higher dimensional sets. Formally, we indicate merge composition with the standard cross-product symbol \times for generating higher dimensional sets. For example, a tablet can be described as the cross product of two one-dimensional tablets, as defined in Equation 6, that have independent axes:

$$\text{GenericTablet}_{xy} = \text{Generic1D Tablet}_x \times \text{Generic1D Tablet}_y,$$

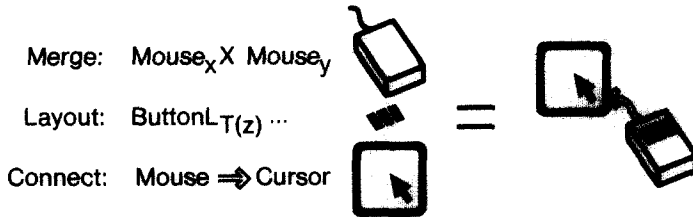
which is shorthand for:

$$\begin{aligned} \text{GenericTablet}_{xy} = \\ & P_{xy}: [Min_x, Max_x] \times [Min_y, Max_y] \\ & -f \rightarrow [f(Min_x), f(Max_x)] \times [f(Min_y), f(Max_y)] \cup \\ & \quad RestState -f \rightarrow Null, \end{aligned}$$

which is, in turn, shorthand for:

$$\begin{aligned} \text{GenericTablet}_{xy} = \\ & < \text{Manipulation:} & P_{xy}, \\ & \text{InputDomain:} & [Min_x, Max_x] \times [Min_y, Max_y] \cup \\ & & < RestState_x, RestState_y >, \\ & \text{State:} & < x, y >, \\ & \text{ResolutionFn:} & f, \\ & \text{OutputDomain:} & [f(Min_x), f(Max_x)] \times \\ & & [f(Min_y), f(Max_y)] \\ & & \cup \\ & & < Null_x, Null_y > \\ & \text{Works:} & \{ \} >. \end{aligned}$$

Figure 7. The combination of a three-button mouse and cursor can be described with the merge, layout, and connect composition operators.



A two-dimensional slider can be described in a similar manner:

$$\text{Generic2DSlider}_{xy} = \text{GenericSlider}_x \times \text{GenericSlider}_y.$$

The notation also suggests other combinations of input device designs, such as a discrete slider crossed with a 1D tablet, which might be appropriate for two related application parameters where one is discrete and the other warrants sensitivity over the entire input range.

The result of connection, layout, or merge composition is still formally an input device. For a connection composition, the composite device is composed of the input domain set of the first device, the output domain set of the second device, and the composition of the two resolution functions. For a layout or a merge composition, the domain sets are combined with a cross product and the resolution functions with a union.

Examples of Composed Input Devices

Mouse. The description of a three-button mouse and cursor combination requires all three composition operators (see Figure 7). The **Mouse** is a composite device that involves a merge composition of two relative position sensors:

$$\text{Mouse} = \text{dP}_{xy}: \text{Real} \times \text{Real}; -f \rightarrow \text{Integer} \times \text{Integer},$$

where dP_{xy} means relative position on the XY plane. The **Mouse3Button** device is described with layout compositions:

$$\begin{aligned} \text{Mouse3Button} = & \\ & \text{Mouse} \otimes \\ & \text{LMouseButton}T_1(z) \otimes \\ & \text{MMouseButton}T_2(z) \otimes \\ & \text{RMouseButton}T_3(z), \end{aligned}$$

where the coordinates of the buttons are relative to the mouse.

The mouse is typically connected (in the sense of a device composition) to a virtual input device called the **Cursor**, which simulates a **2DSlider** device for pointing on the display. The cursor on a bitmapped screen can be described as follows, given that $Screen_x$ and $Screen_y$ are the pixel width and height of the screen:

$$\begin{aligned} \text{Cursor} = \\ dP_{xy}: \text{Integer X Integer} \\ -f \rightarrow \langle 0, \dots, Screen_x \rangle \times \langle 0, \dots, Screen_y \rangle. \end{aligned}$$

The cursor interprets the output of the mouse as relative positional values, updates an internal state description of a location on the screen accordingly, and reports the new location. Because the output of the mouse and the input of the cursor are the same type, their connection typically involves no more than a constant scale factor and clipping, although more elaborate connection functions are sometimes used. Typically, we ignore such issues and write:

$$\text{Mouse} \Rightarrow \text{Cursor}$$

to indicate the cascade from **Mouse** to **Cursor**.

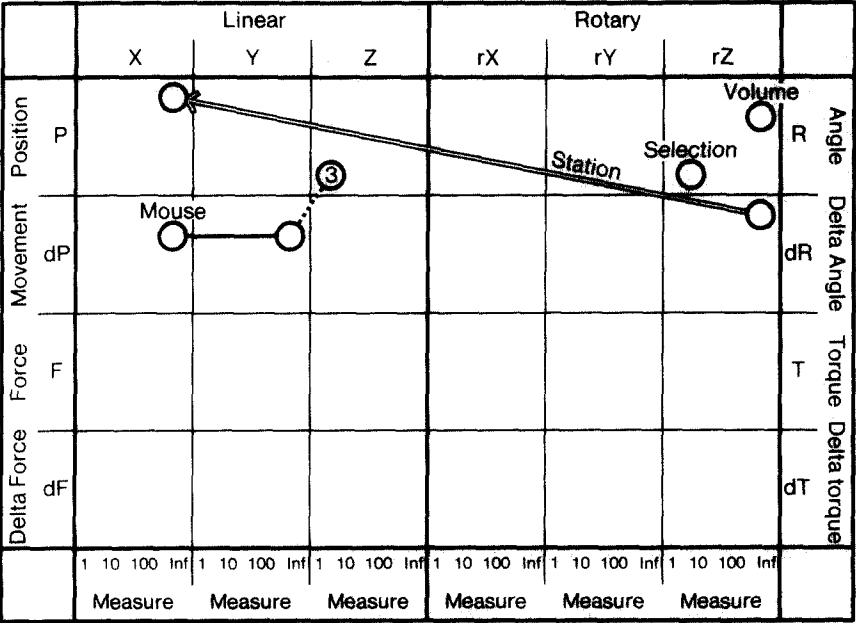
Joystick. A joystick, Joystick_{xy} , is a rotary device that senses rotation around two axes, conventionally the X and Y axes. A Joystick_{xy} can be described as follows:

$$\begin{aligned} \text{Joystick}_{xy} &= 1D \text{ Joystick}_x \times 1D \text{ Joystick}_y \\ 1D \text{ Joystick}_x &= R_x [Min_x \ Max_x] -f \rightarrow [f(Min_x), f(Max_x)]. \end{aligned}$$

An interesting issue raised by comparing joysticks and knobs is whether an input device requires grasping. Knobs often require grasping, whereas joysticks often require only touch. *Graspable* is a physical property of a device that is important for effectiveness evaluation. We can also have a delta joystick with dR_{xy} and a force joystick with T_{xy} , where T stands for torque. We use the works part of our formalism to describe joysticks that spring back to zero rotation.

It should be obvious how to describe 3D versions of some of these devices. One can also describe hybrid versions where one dimension is rotary and the other is linear.

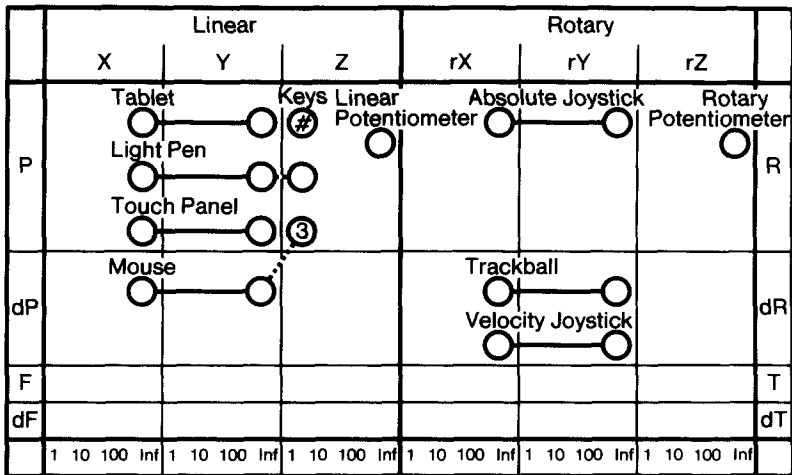
Figure 8. This diagram describes an input device taxonomy that is based on the analysis presented in this article. Circles are used to indicate that a device senses one of the physical properties shown on the vertical axis along one of the linear or rotary dimensions shown on the horizontal axis. For example, the volume circle indicates a device that senses angle around the z axis. The position in a column indicates the number of values that are sensed (i.e., the measure of the domain set). For example, the selection circle represents a discrete device. Lines are used to connect the circles of composite devices. The double line represents a connection composition, such as for the radio station composite device. A black line represents a merge composition, such as the x and y components of the mouse. The dashed line represents a layout composition, such as the buttons on a mouse, which are represented by a circle with a 3 in it (because they are identical devices).



2.6. A Taxonomy of Input Devices

We can now give our equivalent to Foley et al.'s (1984; see Figure 1) and Buxton's (1983; see Figure 2) taxonomies. Our classification is displayed in Figure 8. Devices are classified according to (a) which physical property they transduce, (b) which of the six degrees of freedom they sense, and (c) the measure of the input domain set, which is the number of elements in the domain set. A device is represented in the figure as a set of circles connected together. Each circle represents a transducer in the device, plotted according to the canonical physical property it transduces. Each line indicates a

Figure 9. Foley et al.'s (1984) devices plotted on the taxonomy (see Figure 1).



composition operator: connection composition (double line), layout composition (dotted line), or merge composition (black line).

We have plotted the devices of our radio example and the mouse on this diagram to illustrate its use. The radio volume knob is in the cell in Figure 8 for sensors of angles relative to the Z axis. It is located on the right side of the cell, showing that it is continuous. The selection knob is similar, but it is located nearer the left side showing that it has just a few values. The station knob is located in the cell for relative angle and is connected to a slider for the tuning mechanism. A mouse is depicted in the figure as a circle on X Movement, a circle on Y Movement, and a circle containing the number 3 on Z Positioning. This says that the mouse is a layout composition of four devices: one device that is itself the merge composition of two elementary devices sensing change in X and Y and three other devices, which are simple buttons. The placement of the X and Y circles to the right of the column indicates nearly continuous resolution. The location of the button circles to the left indicates controls with only two states.

We demonstrate the coverage of this taxonomy by showing that it can classify the devices described in the previous taxonomies and by showing that it can classify even unusual devices. Our reclassification of the devices listed by Foley et al. (1984) in Figure 1 is given in Figure 9. The idea of a voice recognizer device, which is considerably more sophisticated than the other devices, is the only one that does not fit in the new taxonomy. Compared with their classification, however, our scheme is more compact, because we have separated the classification of devices from the classification of tasks (which we would include as part of the application), and it is more systematic. We

Figure 10. Buxton's devices plotted on the taxonomy (see Figure 2).

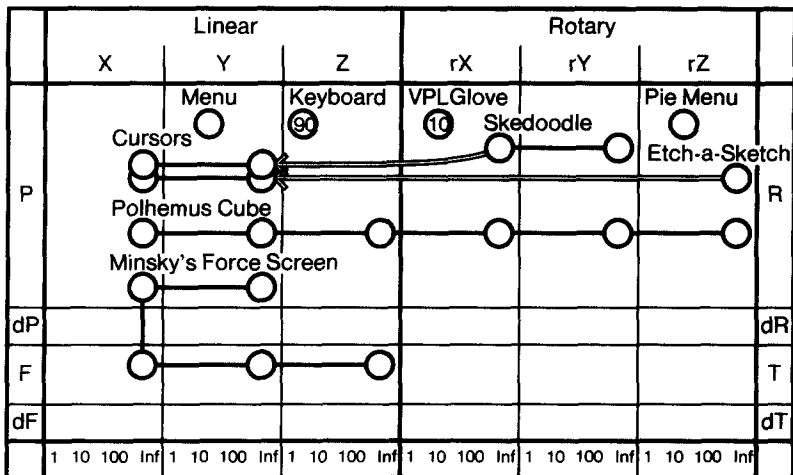
	Linear												Rotary															
	X				Y				Z				rX				rY				rZ							
P					Tablet								Sliding Pot				Absolute Joystick									Rotary Pot	R	
					Light Pen																							
					Touch Panel																							
					Touch Screen																							
dP					Mouse								Treadmill				Trackball								Continous Pot		dR	
					Tasa X-Y Pad								Tasa Ferinstat											Thumbwheel				
F													Pressure Pad				Isometric Joystick								Torque Sensing		T	
dF																											dT	
	1	10	100	Inf	1	10	100	Inf	1	10	100	Inf	1	10	100	Inf	1	10	100	Inf	1	10	100	Inf	1	10	100	Inf

have not, however, attempted in this article to analyze the graphics subtasks or review the performance literature as they did. The figure shows many devices that use absolute or relative linear or rotary position, few that use force or torque, and none that use relative force or torque. Our reclassification of the devices listed by Buxton (see Figure 2) is shown in Figure 10. Our classification improves Buxton's to handle a larger and more systematic set of physical properties, and we can handle both discrete and continuous devices. Except for buttons, all the devices classified by both Buxton's and Foley et al.'s taxonomies were continuous.

Figures 9 and 10 show typical computer input devices. But what about more unusual devices, such as a typewriter keyboard? Figure 11 plots devices that are unusual or extreme in some way.

In addition to the typewriter keyboard, Figure 11 includes the controls for an interesting pair of toys: Etch-a-Sketch and Skedoodle. Buxton (1986) argued that these toys seem "semantically identical," in the sense that they draw using a similar stylus mechanism and even have the same "erase" operator (turn the toy upside down and shake it). But they have very different properties for drawing. Etch-a-Sketch is operated with two knobs, one for *X* and one for *Y*. Skedoodle is operated with a joystick for integrated control of *X* and *Y*. Both of these points come out in Figure 11. Both toys are connection

Figure 11. Some interesting input devices plotted on the taxonomy.



composed to the same sort of cursor device, which we have included in the figure, leading to the appearance that they are semantically identical, but Etch-a-Sketch controls plot as two separate knobs that rotate about Z, whereas the Skedoodle controls plot as two merged controls.

The Polhemus cube is a device that uses magnetic fields to measure the absolute location and orientation of a cube that can be either held or attached to some other object such as a glove or helmet. On the diagram, this control plots as a merge of all the linear and rotary degrees of freedom.

Minsky's (1984) force and position sensitive screen has transducers for position in X and Y but, in addition, can sense force in X, Y, and Z. These transducers allow information on gestures to be captured. The device has a relatively large number of degrees of freedom. Interestingly, this device has transducers for both position in X and Y and force in X and Y.

The VPL glove has transducers to sense the flexing of the fingers. Basically the glove can be seen as a separate set of sensors about an axis of rotation (this depends on the orientation of the hand, but we plot it as if it were the X axis). This gives many degrees of freedom under direct user control. The VPL glove is often composed with the Polhemus cube to provide a large number of degrees of freedom.

Finally, we can plot virtual devices in addition to physical devices. Menus are virtual devices for selecting among alternatives. Most menus would plot as circles in the linear Y position cell. A more exotic sort of menu is a pie menu (Callahan, Hopkins, Weiser, & Shneiderman, 1988). A pie menu is a circular structure divided into slices like a pie (this allows more items to be closer to the starting position of the mouse). The user indicates a selection by moving in a

Figure 12. Sets and their measure.

Set	Measure
Discrete - nominal or ordinal	
{a, b} or <a, b>	2
{a, b, c} or <a, b, c>	3
{a1, a2, ..., an} or <min, ..., max>	Bounded
{a1, a2, ...} or <..., a1, a2, ...>	Unbounded
Continuous	
[min, max]	Bounded
Real	Unbounded

particular direction. The pie menu itself plots onto the diagram in the same place as a rotary switch (assuming eight positions for each). We have not shown it on the diagram, but the pie menu is actually cascaded from the cursor virtual device (which, in turn, is cascaded from the mouse). In this way it is different from a simple rotary switch.

2.7. Evaluating the Expressiveness of an Application's Input Devices

We have said that one of the criteria for the design of artificial languages for human-machine interaction is that they be able to express some intended meaning. Input devices are one of the components used in the human-machine portion of the dialogue, and they make their own contribution to expressiveness.

If we compose the chain of resolution and connection functions from the user to the application, we can compare the semantics of the users manipulations with the desired semantics of the corresponding application parameter. The goal is to satisfy the following principle:

An input device should allow the user to express exactly the information acceptable to the application: neither more nor less.

Formally, the connection among input devices and applications involves the mapping from an **Out** set of an input device to the **In** set of an application or another input device.

One kind of expressiveness problem arises when the measure (Figure 12) of the **Out** set does not match the measure of the **In** set to which it is connected. This leads to a loss of information. If the projection of the **Out** set includes elements that are not in the **In** set, the user can specify illegal values; if the **In**

set includes values that are not in the projection, the user cannot specify legal values.

As an example, consider the **Volume** parameter in our radio example. It is a continuous, bounded domain set, which means the corresponding knob should have a continuous, bounded **Out** set that lets the user specify every decibel value in the **Volume** input domain set. If the **Out** set is discrete or the bounds are too small, the user will not be able to express some decibel levels. If the **Out** set is unbounded or the bounds are too large, the user will be able to express decibel levels outside the range of the radio.

Another sort of expressiveness problem arises if a device has relative semantics, but the application interprets it in an absolute manner. If v is the value and p is the application parameter, an absolute interpretation is:

$$\text{Absolute: } p := v$$

and a relative interpretation is:

$$\text{Relative: } p := p + v.$$

(If nonlinear effects are desired, a function can be wrapped around v .) Input devices, as we have seen, can sense manipulations with either absolute or relative semantics. When an application makes an absolute interpretation of a relative device, there is no simple relationship between a user's manipulation and the application's interpretation. The dual case of absolute device semantics and relative application interpretation is not a problem, because the semantics is simply that of a device controlling a delta or velocity application parameter.

It is interesting to note that applications can also interpret a device value as an acceleration, which can be implemented with p as a special velocity application parameter and p' as the ultimate application parameter to be changed:

$$\text{Accelerated: } p' := p' + p.$$

Velocity and acceleration are interpretations that typically cause users to have strong semantic expectations of the corresponding input device because, in the real world, velocity and acceleration tend to start from zero and vary smoothly. Given this expectation, it is important to check if the input device supports this sort of behavior. For example, a tablet allows sudden changes in its output, but a spring-return joystick always starts from zero and varies smoothly. Clearly, the joystick is the better device for velocity or acceleration parameters.

3. SIMULATED 3D EGOCENTRIC MOTION CONTROLS: ANALYSIS AND GENERATION

We now turn to the use of our theory and taxonomy of input devices in the analysis and innovation of design. Theories and taxonomies often achieve their influence in design as tools for thought. Even formal theories can sometimes be deployed informally to give the designer representations in which to pose and manipulate the design. To this point, we have concentrated on a description of simple, physical input devices, largely in isolation. We now consider the design of more complex virtual input devices in relation to an application. Our application is the design of egocentric motion (movement of the viewpoint) controls as part of a 3D interface. We consider an existing 3D egocentric motion control from the point of view of our theory and taxonomy. We then give a brief case study in which the theory and taxonomy was used to aid in the design of a novel set of movement controls.

3.1. Egocentric Motion

As a user moves about in a simulated 3D world, there is continuous change to the location and gaze of the user's viewpoint relative to an origin maintained at the point of view. The six usual degrees of freedom of position and orientation lead to 12 application parameters:

Body position

Move-Right	Move-Left
Move-Up	Move-Down
Move-Forward	Move-Back

Body orientation

Rotate-Right	Rotate-Left
Rotate-Up	Rotate-Down
Rotate-Clockwise	Rotate-Counterclockwise.

When a coordinate axis is placed at the point of view origin, these 12 parameters can be reduced to six parameters by combining the parameters that have the obvious inverse relationships:

Body position

Move-Right-Left
Move-Up-Down
Move-Forward-Back

*Body orientation***Rotate-Right-Left****Rotate-Up-Down****Rotate-Clockwise-Counterclockwise.**

We use the convention that **Move-Right-Left** means that positive values indicate how far to move right and negative values indicate how far to move left.

Although control of six degrees of freedom is sufficient for specifying any position and orientation of the viewpoint, more degrees of freedom can be used to support a metaphor of walking, thereby allowing the user to transfer to the virtual world movement techniques from the real world. In this metaphor, there are two coordinate axes at the point of view, one for the body and one for the gaze. Of the six body parameters, functionality associated with the body coordinates is typically limited to the position parameters and the left-right rotate parameters. Of the six gaze parameters, functionality is typically limited to four—The parameters in parentheses are not used:

*Gaze orientation***Gaze-Right****Gaze-Left****Gaze-Up****Gaze-Down****(Gaze-Clockwise) (Gaze-Counterclockwise).**

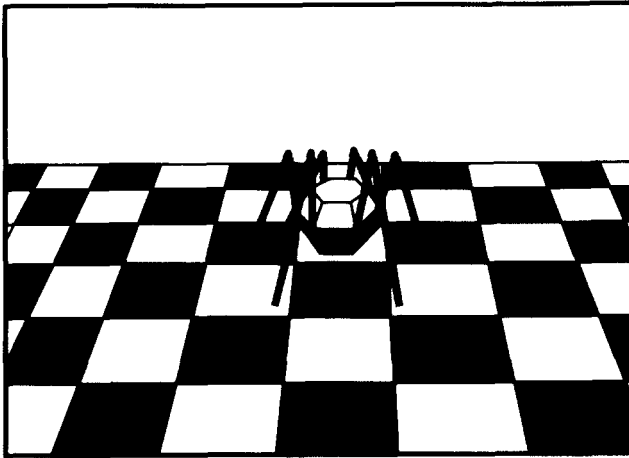
Like any application, an egocentric motion application can interpret values from input devices as absolute change, relative change (which can be thought of as velocity), or acceleration change to its application parameters. The most common choice for egocentric motion application is relative change, which is the interpretation assumed in the following application descriptions unless otherwise stated.

3.2. Analysis of an Existing Egocentric Motion Design

An example of input controls for egocentric motion is based on the demonstration program *Insect*, written by Thant Tessman from Silicon Graphics, Inc. (see Figure 13). The insect is impressively animated and walks about more or less under control of the user. The user can move about to view the insect from different places. It gives an elegant demonstration of the interactive graphics capabilities of the Iris series of computers.

Insect uses three physical input devices for egocentric motion: the mouse, the Control key, and the Left Shift key. The virtual devices are the cursor connected to the mouse and two instantiations of the following generic device, which can be characterized as a window-centered tablet device:

Figure 13. Tessman's Insect demo has an insect walking on a checkerboard grid.



WinTablet =

$$\begin{aligned}
 P_{xy}: & \langle 0, \dots, Win_x \rangle \times \langle 0, \dots, Win_y \rangle \\
 -f \rightarrow & \langle -Win_x/2, \dots, +Win_x/2 \rangle \times \\
 & \langle -Win_y/2, \dots, +Win_y/2 \rangle,
 \end{aligned}$$

where the device is sensitive over its input range and springs back to zero when not in use. Win_x and Win_y represent the width and height of the Insect window and the function f maps from window coordinates to window-centered coordinates. Although we call this device a tablet because of its input range sensitivity, it is also like a joystick in that the output state is reset to zero when the user stops manipulating it. Virtual devices often blend the properties of common physical devices.

The left mouse button activates one instantiation **LMWinTablet**, and the middle mouse button activates the other **MMWinTablet**. Formally, these activations can be modeled as connections from the mouse buttons to $\{ON, OFF\}$ inputs that have been added to the corresponding virtual tablet descriptions (see the Appendix for details). The insect virtual controls are plotted on our taxonomy diagram in Figure 14. The first thing to notice is that the virtual controls for these applications become quite complex. Figure 15 describes the connections to the application parameters. Such diagrams help us to keep track of the relationships among the composed input devices.

We evaluate the expressiveness of the Insect egocentric motion controls by following the control's resolution and connection mappings from the physical device manipulations to the application parameters, which results in the following table:

Figure 14. The composite device for the Insect demo plotted on the taxonomy.

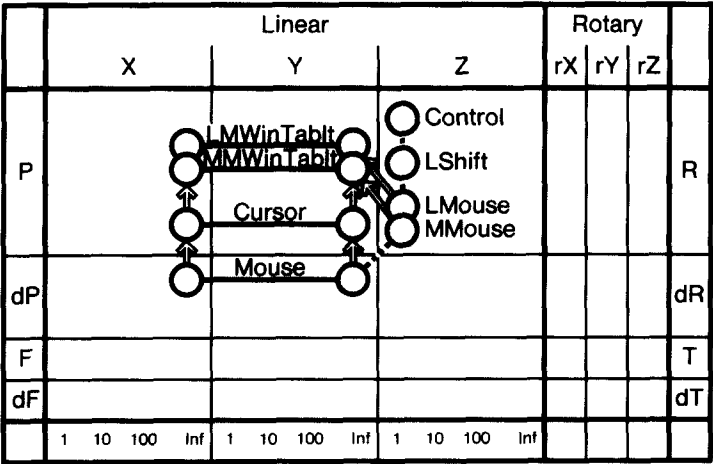
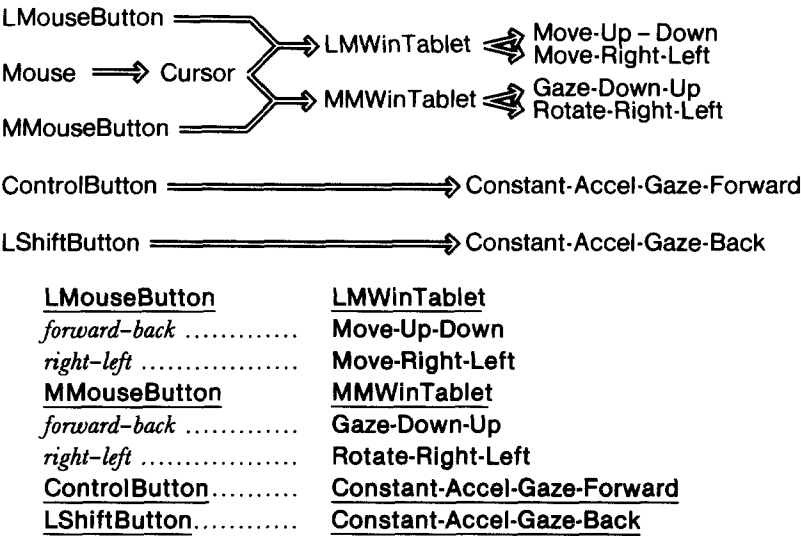


Figure 15. The devices and connections for the movement controls of the Insect demo.



This table can be read as follows: The left mouse button **LMouseButton** turns on **LMWinTablet** mode: Moving the mouse forward or back moves the viewpoint up or down by an amount corresponding to the distance of the mouse from the center of the window. The other entries in the table are read similarly. The **ControlButton** causes the viewpoint to be accelerated a

constant amount forward along the axis of the gaze and similarly for the **LShiftButton**. Although not indicated in the table, movement stops when the keys or buttons are released; **LShiftButton** input dominates the **ControlButton** when both are pressed.

The right side of the table shows that the **Insect** controls allow the user to express the basic functionality of walking around a 3D space. Our analysis, however, suggests some of the reasons why even experienced users have difficulties with the **Insect** movement controls.

The first problem has to do with the use of tablets. Because tablets are sensitive over their entire input range, abrupt movements are possible, which can be distracting and confusing. Inasmuch as the semantics of these tables are velocity parameters, it is particularly problematic that the **Insect** uses virtual tablets.

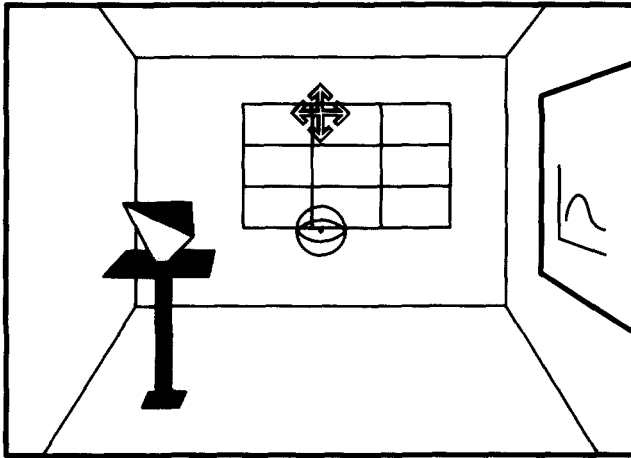
The second problem has to do with the **MMouseButton** control of rotating the gaze down and up. The semantics of this control is not consistent with the other **MMouseButton** control of rotating the body right and left (it is reversed). It is also inconsistent with the **LMouseButton** control of body movement up and down (it is also reversed). It is possible that the designer of this device reversed this control so that the combined operation of **LMouseButton** and **MMouseButton** up and down would give a certain effect (moving the body up while looking down or moving down while looking up). However, the more common uncombined operations are much harder to use because of the inconsistencies. In addition, the mixture of gaze rotation and body rotation on the same control (middle mouse) can be confusing.

A third problem is the lack of limits on movement. For example, the gaze can be turned 180° and leave the user looking upside down. Another example is that the user can rotate either the body or the gaze so that the plane of interest (the checkerboard that the insect walks on) is out of sight. When that happens, it is very easy for the user to get totally lost.

3.3. Generation of Novel 3D Movement Controls: 3D Rooms

In the previous section, our theory of input devices was used to comprehend and critique the design of an existing virtual input control. In this section, we sketch a short case study of the use of the theory for designing novel input controls for egocentric motion in a 3D world as part of the design of a prototype for a 3D version of *Rooms* (Henderson & Card, 1986; Robertson, Card, & Mackinlay, 1989; see Figure 16). The design steps of the case study proceeded roughly as follows: (a) identify application functionality, (b) assign input devices to the application's parameters in a manner consistent with the expressiveness criteria, and (c) compose these devices together in a manner consistent with the expressiveness criteria. This case study does not include

Figure 16. An exploratory room in the 3D Rooms system. Two movement icons are presented to the user in a heads-up display. The arrow icon is used to move in the plane of the body. The eye icon is used to rotate the gaze. The user can specify walking movement by pointing elsewhere in the window.



the formal evaluation of effectiveness criteria, which might require an additional design step to test the completed design (Mackinlay, 1986a, 1986b).

Step 1: Application Functionality

The first step was to identify the application functionality to be controlled by the user. We decided to provide movement functionality based on the metaphor of human movement around a room. To carry out this metaphor, the user is placed in an environment matched to human scale, and the viewer corresponds to human body orientation. In the default position, the user's eyes are 5' 4" above the floor, looking forward and slightly down. We identified the following six application parameters associated with the walking metaphor:

Body

- Move-Right-Left
- Move-Up-Down
- Move-Forward-Back
- Rotate-Right-Left

Gaze

- Gaze-Right-Left
- Gaze-Up-Down

Body constraints are mirrored in movement constraints: (a) body position is restricted by the room's walls, and (b) gaze orientation is restricted to angles that are natural for the human head. Additional application functionality was added for common movements such as standing up and centering the gaze.

Step 2: Choosing Input Devices

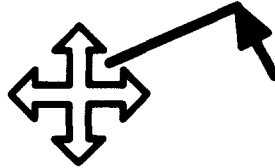
The next step was to choose appropriate input devices for each parameter. Expressiveness suggests that an important variable in this choice is the measure of the ranges of the various application parameters. Again, in line with the metaphor of walking, some ranges are fixed and limited, such as the gaze orientation; some ranges are variable, such as the sizes of different rooms; and some ranges are unbounded, such as body rotation. Fixed limited ranges can be assigned to input devices with connections that support a direct correspondence between the input device state and the state of the application parameter. For example, gaze orientation could be assigned to an absolute joystick where the position of the stick corresponds directly to the position of the gaze. Variable and unbounded ranges, on the other hand, require a different design. A good design is to have the application interpret the input value as a relative change and use an input device that has properties similar to a joystick—a range from negative to positive and a spring back to zero when the user stops manipulating the device. The Insect controls use this type of design. We decided to use the relative interpretation uniformly for the six walking application parameters even though the gaze could be handled in a different way, because our physical input devices consisted of a keyboard and a mouse. A two-dimensional virtual device was developed that converts screen coordinates to joystick style coordinates:

VirtualJoystick =

$$P_{xy}: \langle 0, \dots, Screen_x \rangle \times \langle 0, \dots, Screen_y \rangle \\ -f \mapsto \langle Min_x, \dots, Max_x \rangle \times \langle Min_y, \dots, Max_y \rangle.$$

The zero location of the virtual joystick is determined at instantiation time, which occurs when the **MMouseButton** is pushed. In this way, the other mouse buttons are freed for other application functionality, and specific virtual joysticks can be tied to icons. The user moves the **Cursor** over a transparent movement icon, pushes the **MMouseButton**, and begins manipulating a virtual joystick device positioned at the cursor. Although these virtual devices are not technically joysticks in that they do not sense rotation, they are similar to physical joysticks, and we use the graphical feedback shown in Figure 17 that suggests a joystick device. The mapping function f has a flat spot around zero to keep the control from being too sensitive.

Figure 17. Joystick feedback is provided by drawing a line from the current mouse position, as indicated by the mouse cursor, to the cursor position at the time the joystick was instantiated. In this case, the user is moving in the plane of the body.



Step 3: Designing Composite Input Controls

Given choices of input devices for the individual application parameters, the next step was to design a composite control, mindful of the application semantics. The merge composition of the two gaze parameters into a single joystick was obvious. The connect of left-right, forward-back, and up-down to mouse movements to the corresponding directions on the mouse was also obvious and fairly conventional. The only remaining question was, should **Move-Forward-Back** be merged with **Rotate-Right-Left** or **Move-Right-Left**. We tried both and found that the tanklike **Rotate-Right-Left** combination worked better. The **Move-Right-Left** combination is not a common walking movement. Therefore, our design has three virtual joystick devices. Two of these are tied to icons and the third virtual joystick (the most common) is instantiated whenever the middle mouse button is pressed when the cursor is not over one of the movement icons (Figure 18). The result is the following expressiveness relationship between virtual joystick devices and mouse movement:

<u>MMouseButton:</u>	<u>3D Rooms Navigation</u>
<u>No icon:</u>	<u>TankVirtJoy</u>
<i>forward-back</i>	Move-Forward-Back
<i>right-left</i>	Rotate-Right-Left
<u>Arrows icon:</u>	<u>BodyVirtJoy</u>
<i>forward-back</i>	Move-Up-Down
<i>right-left</i>	Move-Right-Left
<u>Eye icon:</u>	<u>GazeVirtJoy</u>
<i>forward-back</i>	Gaze-Up-Down
<i>right-left</i>	Gaze-Right-Left
<u>Click on icon</u>	<u>StandUp</u> and <u>CenterGaze</u>

The virtual joysticks are also sensitive to quick clicks of the mouse button so that the user can stand back up to the default height and center the gaze. The

Figure 18. The composite device for our 3D Rooms application plotted on the taxonomy.

	Linear												Rotary			
	X				Y				Z				rX	rY	rZ	
P																R
dP																dR
F																T
dF																dT
	1	10	100	Inf	1	10	100	Inf	1	10	100	Inf				

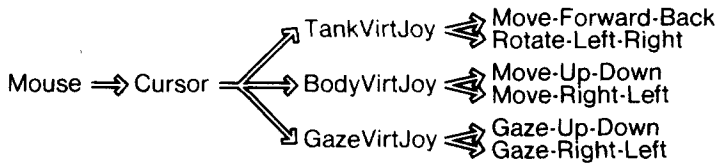
resulting connectivity for the input devices of 3D rooms is given in Figure 19. In this case, the theory was used as tools of thought for conceptualizing the design space. Part of the actual process of design consisted of proposing alternative composition operators by writing down short forms of the notation on the whiteboard.

4. DISCUSSION

Let us summarize our semantic analysis of the design space of input devices. We began with Buxton's (1983) notion that input devices are transducers of physical properties in one, two, or three dimensions. We augmented his set of three elementary physical properties—position, motion, and pressure—to include all eight possible combinations of linear and rotary, absolute and relative, position and force (Figure 5). Although exotic input devices might be imagined that transduce other physical properties (e.g., galvanic skin response), and some input devices may require very complex signal processing (e.g., speech recognition or hand-print recognition), this seems to be the basic, canonical set of physical properties available for sensing and communication with the machine.

We also augmented Buxton's description of space. To his three linear dimensions, we added three rotational dimensions (and relative values—essentially the first derivative with respect to time). We traced the mappings of the physical transducer through the device and into the semantics of the application. One of these mappings, the resolution function, allowed us to go

Figure 19. The devices and connections for the movement controls for our 3D Rooms application.



beyond the continuous devices Buxton was able to describe to discrete devices and, in fact, devices in between.

In order to be able to describe a wide range of devices, we deployed two basic combinatoric techniques. One was to define hierarchical families of generic devices, and the other was to define composition operators. Both allowed the description of more complex devices in terms of simpler devices. In the analysis, the object was to give an account of a broad range of different input device designs—including discrete, composite, physical, and virtual devices—and to do so in terms of a small set of primitive and combinatorial operations. These combinatoric techniques can also be used as technical abstractions to quickly explore design alternatives before the details of a specific design must be decided.

With this description in place, we exercised it in several ways. One was to produce a simple taxonomy of input devices. We compared that taxonomy with other taxonomies by showing that we could handle those devices previously classified by other systems as well as new devices not previously classified, including some that are mildly exotic. In another exercise, we applied our description to the design of virtual input controls in a frontier area of user interface technology—the design of egocentric motion controllers for virtual 3D worlds. There we used our techniques to help us understand and critique an existing controller. We then used them to help design a new controller of our own.

The semantic analysis and taxonomy described in this article suggest a number of directions for future work on input devices and user interfaces.

One direction is the development of user interface toolkit software. For example, Anson's (1982) device model results in a particularly receptive toolkit architecture. His toolkit uses a PASCAL-like notation to specify simple devices, which consist of (a) state, (b) events, and (c) actions, and composite devices, which are devices connected together by CHANNEL statements. A device's actions are sensitive to events and the states of devices. When an action triggers, it can change the device state and signal events to other devices. This architecture could be used to implement a library of useful input devices using concepts developed in our analysis. Resolution functions could be implemented by device actions that convert input states to output states.

The physical properties of our taxonomy itemize the types of input states that must be supported by the library. Our generic devices could be supported by extending Anson's programming language in an object-oriented manner. Input device composition could be implemented with this object-oriented extension and his CHANNEL statements. In short, our analysis and a basic toolkit architecture could be used to develop a useful library of modules for designing input devices.

Another direction is suggested by Mackinlay's (1986a, 1986b) work on graphical presentation, which motivated the approach taken in this article. Mackinlay identified expressiveness and effectiveness criteria for evaluating the designs of graphical presentations. In this article, we have concentrated on the expressiveness of input devices, which allowed us to understand more about the design space of input devices. Research on input device effectiveness addresses the human science issues associated with designing input controls, such as how devices fit the shape of the hand and how easy devices are to manipulate within given ranges. The ultimate goal is to devise effectiveness criteria that allow the direct comparison of alternatives from the design space of input devices. This can be very useful. Given such effectiveness criteria for graphical presentation, Mackinlay used artificial intelligence techniques to develop a program called "A Presentation Tool" (APT), which automatically designed graphical presentations of information. APT used primitive graphical languages, composition operators, and various evaluation criteria to generate and test alternative designs of graphical presentations. The semantic analysis described in this article develops the primitive and composition operators to generate alternative designs of input devices. Given appropriate effectiveness criteria for input devices, it should be possible to build systems that assist in the design of an application's input controls or even design them automatically.

An input device is part of the means used to engage in dialogue with a computer or another machine. There are, of course, other components that must be described. We have treated input device actions as sentences in a simple artificial language for conveying meaning from a human into a machine. These simple artificial languages can be composed to be parts of more complex artificial languages. Our study began with very simple single-transducer languages. These were composed into multitransducer languages capable of more complex expression. In Mackinlay's (1986a, 1986b) study, he went in the opposite direction and analyzed machine to human communication. Thus we have two studies, using similar techniques, one going in each direction. We are mindful of the many steps to go in the analysis of human-machine interaction: a treatment of time, of feedback, of discourse control structure, and many other topics. Yet, although this is scientifically daunting, the task is mitigated by the likelihood that useful engineering techniques will occur with each step.

5. CONCLUSION

Computer science has always been a curiously asymmetrical discipline, with a large amount of work on the machine side, but little on the human side, despite the fact that many computer science systems areas (e.g., programming languages, operating systems, software engineering, or computer graphics) depend critically on properties of human performance. Partially, we think, this is because it has been difficult to get a common technical framework for computational and human sciences (Newell & Card, 1985). The approach used in this article has the virtue that each of these has a well-defined complementary role. Advances on either side can be exploited: We can exploit advances in the technical representation of diagrams or advances in unified theories of cognition (Newell, 1987).

Work on these foundational issues of user interface component semantics can lead to several practical outcomes. First, technical abstractions about the user interface can support the design of effective user interfaces and effective user interface toolkits. Second, there is the possibility of user-situated designs, designs done at the time of need. Users may be able to use theory-based toolkits directly, or agents might be developed that automatically adjust the interface design to the needs of the user. Third, combinatorial complexity of application functionality is one of the fundamental problems of user interface design. The ability to analyze semantic properties of the interface allows the technical design of consistency for the interface, and it allows the control of combinatoric functionality with interface features. Fourth, a user interface design developed for the basic functionality of an application rarely supports the incorporation of new objects and operators as application functionality grows. A technical model of interface semantic properties could allow the construction of user interfaces capable of at least limited self-adjustment. But, most important, improvements in the technical foundations of interface design should help in the systematization of the body of engineering and design knowledge for human-machine interfaces.

Acknowledgments. We thank Allen Newell, an anonymous reviewer, and Peter Polson, our editor, for their suggestions, which helped us in many ways including clarifying the focus of this article.

REFERENCES

- Anson, E. (1982). The device model of interaction. *Computer Graphics*, 16(3), 107-114.
- Baecker, R. M., & Buxton, W. (Eds.). (1987). *Readings in human-computer interaction: A multidisciplinary approach*. Los Altos, CA: Morgan Kaufmann.
- Bertin, J. (1983). *Semiology of graphics* (W. J. Berg, Trans.). Milwaukee: University of

- Wisconsin Press. (Original work published 1967)
- Buxton, W. (1983). Lexical and pragmatic considerations of input structures. *Computer Graphics*, 17(1), 31-37.
- Buxton, W. (1986). There's more to interaction than meets the eye: Some issues in manual input. In D. A. Norman & S. W. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction* (pp. 319-337). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Callahan, J., Hopkins, D., Weiser, M., & Shneiderman, B. (1988). An empirical comparison of pie vs. linear menus. *Proceedings of the CHI '88 Conference on Human Factors in Computing Systems*, 95-100. New York: ACM.
- Card, S. K., Mackinlay, J. D., & Robertson, G. G. (1990). The design space of input devices. *Proceedings of the CHI '90 Conference on Human Factors in Computing Systems*, 117-124. New York: ACM.
- Foley, J. D., Wallace, V. L., & Chan, P. (1984). The human factors of computer graphics interaction techniques. *IEEE Computer Graphics & Applications*, 4(11), 13-48.
- Henderson, D. A., Jr., & Card, S. K. (1986). Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics*, 5(3), 211-243.
- Mackinlay, J. (1986a). *Automatic design of graphical presentations*. Unpublished doctoral dissertation. Stanford University, Computer Science Department, Stanford, CA. Also Tech. Rep. Stan-CS-86-1038.
- Mackinlay, J. (1986b). Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2), 110-141.
- Mackinlay, J. (1988). Applying a theory of graphical presentation to the graphic design of user interfaces. *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, 179-189. New York: ACM.
- Mackinlay, J., & Genesereth, M. R. (1985). Expressiveness and language choice. *Data Knowledge Engineering*, 1(1), 17-29.
- Minsky, M. (1984). Manipulating simulated objects with real-world gestures using a force and position sensitive screen. *Computer Graphics*, 18(3), 195-203.
- McCormack, J., & Asente, P. (1988). An overview of the X Toolkit. *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, 46-55. New York: ACM.
- Newell, A., & Card, S. (1985). The prospects for psychological science in human-computer interaction. *Human-Computer Interaction*, 1, 209-242.
- Newell, A., & Card, S. (1986). Straightening out softening up: Response to Carroll and Campbell. *Human-Computer Interaction*, 2, 251-267.
- Newell, A. (1987). *Unified theories of cognition: The William James Lectures* [Video]. Cambridge, MA: Harvard University, Department of Psychology.
- Olsen, D. R., Jr., Kasik, D., Rhyne, J., & Thomas, J. (1987). ACM SIGGRAPH workshop on software tools for user interface management. *Computer Graphics*, 21(2), 71-147.
- Olsen, D. R., & Halversen, B. W. (1988). Interface usage measurements in a user interface management system. *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, 102-108. New York: ACM.
- Pfaff, G. E. (1985). *User interface management systems*. New York: Springer-Verlag.
- Robertson, G. G., Card, S. K., & Mackinlay, J. (1989). The cognitive coprocessor architecture for interactive user interfaces. *Proceedings of ACM Symposium on User*

- Interface Software & Technology*, 10-18. New York: ACM.
- Sherr, S. (1988). *Input devices*. Boston: Academic.
- Tanner, P. P., & Buxton, W. A. S. (1985). Some issues in future UIMS development. In G. E. Pfaff (Ed.), *User interface management systems* (pp. 67-79). New York: Springer-Verlag.
- Tessman, T. (1988). *Insect* [Computer program]. Mountain View, CA: Silicon Graphics, Inc.
- van den Bos, J. (1988). Abstract interaction tools: A language for user interface management systems. *ACM Transactions on Programming Languages and Systems*, 10(2), 215-247.

HCI Editorial Record. First manuscript received April 3, 1989. Revision received June 5, 1989. Final manuscript received September 7, 1989. Accepted by Peter Polson. — *Editor*

APPENDIX. DETAILED DESCRIPTION OF INSECT CONTROLS

This appendix shows that the formalism developed in this article can be used to describe the complex input controls of a real application, namely, the Insect demo written by Thant Tessman from Silicon Graphics, Inc. We describe the application parameters of the Insect demo, the basic physical and virtual input devices used in the Insect demo, the connections among these devices, and the resulting composite input device.

Application Parameters

The following six application parameters describe the egocentric motion functionality of the Insect demo:

Move-Up-Down
Move-Right-Left
Gaze-Down-Up
Rotate-Right-Left
Constant-Accel-Gaze-Forward
Constant-Accel-Gaze-Back

Formally, these parameters represent devices (not input devices) that are part of the Insect application.

For example, the application device **Move-Up-Down** interprets input values as a vertical distance to be moved relative to the *Y* axis position of the body. The resolution function *f* of the **Move-Up-Down** device depends both on the value to move and on the current body position:

Move-Up-Down =

$$\text{MoveUD} \times \text{Body}_y \xrightarrow{-f} \text{Body}_y,$$

where $f(\text{MoveUD}, \text{Body}_y) = \text{Body}_y + \text{MoveUD}$.

In this expression, the domain sets **MoveUD** and **Body_y** refer respectively to the distance to move up or down and to the *Y* axis position of the body. *MoveUD* and *Body_y* refer to the corresponding state variables of the device. Because the resolution function *f* maps to the domain set **Body_y**, the corresponding state variable *Body_y* is changed when *f* is computed by the device. **Move-Right-Left**, **Gaze-Down-Up**, and **Rotate-Right-Left** can be described in a similar manner.

The application device **Constant-Accel-Gaze-Forward** interprets its value as a switch that causes a constant acceleration relative to the *Z* axis of the gaze coordinate system. Because acceleration is involved, we need the internal state variable, *velocity*, to hold the current velocity. Given the internal state variable, *Gaze_z*, for the body position relative to the gaze's *Z* axis, the following describes the **Constant-Accel-Gaze-Forward** device:

Constant-Accel-Gaze-Forward =

$$\text{Switch} \times \text{Gaze}_z \times \text{Velocity} \xrightarrow{-f} \text{Gaze}_z \times \text{Velocity},$$

where $f(\text{Switch}, \text{Gaze}_z, \text{Velocity}) =$

$$\begin{aligned} (\text{Switch} = \text{On}) &\rightarrow \langle \text{Gaze}_z + \text{Velocity}, \\ &\quad \text{Velocity} + \text{Const}_{\text{Accel}} \rangle \\ (\text{Switch} = \text{Off}) &\rightarrow \langle \text{Gaze}_z + \text{Velocity}, \\ &\quad \max(0, \text{Velocity} - \text{Const}_{\text{Decel}}) \rangle. \end{aligned}$$

When the switch is on a constant acceleration, *Const_{Accel}* is added to the velocity; when the switch is off a constant deceleration, *Const_{Decel}* is subtracted until the velocity returns to zero. As we shall see, the **Switch** input to this device is connected to the output of the Control key, which is a button device. **Constant-Accel-Gaze-Back** can be described in a similar manner, where the velocity is subtracted rather than added.

Basic Input Devices

Five physical and three virtual input devices can be used to describe the egocentric motion control of the Insect demo. The generic input device:

$$\text{GenericButton}_z = P_z: [0, \text{Max}_z] \xrightarrow{-f} \langle \text{Up}, \text{Down} \rangle$$

can be used to describe four of the Insect's physical input devices: **LMouseButton**, **MMouseButton**, **ControlButton**, and **LShiftButton**, which are the left and middle mouse buttons and the Control and Left Shift keys. For example, the left mouse button can be instantiated as follows:

$$\text{LMouseButton}_{T_{(c)}} = \text{Instantiate}(\text{GenericButton}_z, \text{In: } [0, \text{Max}_{T_{(c)}}]).$$

The final physical input device is the **Mouse** device, which has already been described in the body of this article. The **Mouse** is connected to the **Cursor** virtual input device, which has also already been described.

The other two virtual devices are the two window-based tablets: **LMWinTablet** and **MMWinTablet**. Although these tablets are based on the **WinTablet** device that is described in the text, the two window tablets have an additional input connected to the appropriate mouse button that is used to activate the tablet. The following describes the generic switched window tablet:

$$\begin{aligned} \text{SwitchedWinTablet} = \\ P_{xy}: <0, \dots, \text{Win}_x> \times <0, \dots, \text{Win}_y> \oplus \text{Switch} \\ \quad -f \mapsto < -\text{Win}_x/2, \dots, +\text{Win}_x/2> \times \\ \quad \quad < -\text{Win}_y/2, \dots, +\text{Win}_y/2> \\ \text{where } f(X, Y, \text{Switch}) = \\ \quad (\text{Switch} = \text{Off}) \rightarrow <0, 0> \\ \quad (\text{Switch} = \text{On}) \rightarrow <X - \text{Win}_x/2, Y - \text{Win}_y/2>. \end{aligned}$$

The values Win_x and Win_y refer to the width and height of the Insect window. The tablet returns zero when the switch is off and converts the input to window centered coordinates when the switch is on. The following instantiations describe the tablets used in the Insect demo:

$$\text{LMWinTablet} = \text{Instantiate}(\text{SwitchedWinTablet}, \text{Switch: LMSwitch})$$

$$\text{MMWinTablet} = \text{Instantiate}(\text{SwitchedWinTablet}, \text{Switch: MMSwitch}),$$

where each tablet has a unique domain set for its switch.

Connections

Given a description of the application parameters and basic input devices of the Insect demo, the next step is to describe how these devices are connected. In this article, we used the ternary relation *Connect* to describe the mapping from the output domain set of one device to the input domain set of another, which is sufficient for simple connections. When dealing with composite devices and real applications, however, the connections can easily be among subdimensions of the domain sets. For example, the connection from the three-button mouse to the cursor device is from the subdimension of the mouse sensor to the cursor input domain set. Therefore, we extend our notation for connection devices to include a subscript of the corresponding device's name of the subdimension. For example, the connection between the

control button and the forward acceleration application parameters can be described as follows:

$$\begin{aligned}
 \text{CAGF} &= \text{Connect}(\text{ControlButton}_{\text{Out}}, \\
 &\quad \text{Constant-Accel-Gaze-Forward}_{\text{Switch}}, \\
 f(\text{ControlButton}_{\text{Out}}) &= \\
 (\text{ControlButton}_{\text{Out}} = \text{Up}) &\rightarrow \langle \text{Off} \rangle \\
 (\text{ControlButton}_{\text{Out}} = \text{Down}) &\rightarrow \langle \text{On} \rangle.
 \end{aligned}$$

The subscripted **ControlButton_{Out}** refers to entire output domain set of the control button, and **Constant-Accel-Gaze-Forward_{Switch}** refers to the switch part of the corresponding application device. The connection function converts the *Up*, *Down* of the button to the *<Off, On>* of the switch domain set. We name this connection device with the acronym **CAGF**, which is based on the connection's output domain set. We use similar mnemonics for the other connection to reduce the length of the description of the composite insect device. The connection **CAGB** from **LShiftButton_{Out}** to **Constant-Accel-Gaze-Back_{Switch}** can be described in a similar manner.

The connection from the mouse sensor to the cursor uses the identity function:

$$\text{CM} = \text{Connect}(\text{ThreeButtonMouse}_{\text{Sensor}}, \text{Cursor}_{\text{In/I}}),$$

where **ThreeButtonMouse_{Sensor}** = **Cursor_{In}** = Integer X Integer.

The connection from the cursor to the left mouse window tablet is only defined when the cursor is over the window. If the window is bounded by *<Min_x, Min_y>* and *<Max_x, Max_y>*, the connection converts the cursor position from screen coordinates to window coordinates:

$$\begin{aligned}
 \text{LWT}_{\text{XY}} &= \text{Connect}(\text{Cursor}_{\text{Out}}, \text{LMWinTablet}_{\text{XY}}, \\
 f(X, Y) &= (\langle \text{Min}_x, \dots, \text{Max}_x \rangle \times \langle \text{Min}_y, \dots, \text{Max}_y \rangle) - \\
 &\quad \langle X - \text{Min}_x, Y - \text{Min}_y \rangle).
 \end{aligned}$$

The connection **MWT_{XY}** from the cursor to **MMWinTablet_{XY}** can be described in a similar manner.

The connection from the left mouse button to the corresponding window tablet switch can be described as follows:

$$\begin{aligned}
 \text{LWT}_s &= \text{Connect}(\text{LMouseButton}_{\text{Out}}, \text{LMWinTablet}_{\text{Switch}}, \\
 f(\text{LMouseButton}_{\text{Out}}) &= \\
 (\text{LMouseButton}_{\text{Out}} = \text{Up}) &\rightarrow \langle \text{Off} \rangle \\
 (\text{LMouseButton}_{\text{Out}} = \text{Down}) &\rightarrow \langle \text{On} \rangle).
 \end{aligned}$$

The connection MWT_s can be described in a similar manner.

The two virtual tablets are connected to four application parameters, and these connections use constants to scale from window coordinates to the appropriate distances to be moved:

$$\begin{aligned}
 MUD &= \text{Connect}(\text{LMWinTablet}_Y, \text{Move-Up-Down}_{\text{MoveUD}}, \\
 &\quad f(Y) = Y \times \text{MUDConstant}), \\
 MRL &= \text{Connect}(\text{LMWinTablet}_X, \text{Move-Right-Left}_{\text{MoveRL}}, \\
 &\quad f(X) = X \times \text{MRLConstant}), \\
 GDU &= \text{Connect}(\text{MMWinTablet}_Y, \text{Gaze-Down-Up}_{\text{GazeDU}}, \\
 &\quad f(Y) = Y \times \text{GDUConstant}), \\
 RRL &= \text{Connect}(\text{MMWinTablet}_X, \text{Rotate-Right-Left}_{\text{RotRL}}, \\
 &\quad f(X) = X \times \text{RRLConstant}).
 \end{aligned}$$

Composite Insect Device

Finally, we describe the composite insect input device, which maps from the physical properties that the user can manipulate to the appropriate application parameters. This description takes advantage of the functional formalism developed in this article, which allows us to use the function composition operator \circ to combine devices. The input domain set is the cross product of the mouse cursor and four buttons for the two mouse buttons and the two keyboard keys. We use the constant D to represent the maximum depth of a button press and the constant C to represent the threshold of a button press after which the button is considered to be down:

$$\begin{aligned}
 \text{INSECT} &= \\
 &\quad \text{Real } X \text{ Real } \otimes [0, D] \otimes [0, D] \otimes [0, D] \otimes [0, D] \\
 &\quad \xrightarrow{-f-} \\
 &\quad \text{Move-Up-Down}_{\text{MoveUD}} \ X \\
 &\quad \text{Move-Right-Left}_{\text{MoveRL}} \ X \\
 &\quad \text{Gaze-Down-Up}_{\text{GazeDU}} \ X \\
 &\quad \text{Rotate-Right-Left}_{\text{RotRL}} \ X \\
 &\quad \text{Constant-Accel-Gaze-Forward}_{\text{Switch}} \ X \\
 &\quad \text{Constant-Accel-Gaze-Back}_{\text{Switch}} \\
 &\text{where } f(X, Y, LMB, MMB, CB, LSB) = \\
 &\quad < MUD \circ \text{LMWinTablet}(\\
 &\quad \quad \text{LWT}_{XY} \circ \text{Cursor} \circ \text{CM} \circ \text{Mouse}(X, Y), \\
 &\quad \quad \text{LWT}_s \circ \text{LMouseButton}(LMB)), \\
 &\quad MRL \circ \text{LMWinTablet}(\\
 &\quad \quad \text{LWT}_{XY} \circ \text{Cursor} \circ \text{CM} \circ \text{Mouse}(X, Y), \\
 &\quad \quad \text{LWT}_s \circ \text{LMouseButton}(LMB)), \\
 &\quad GDU \circ \text{MMWinTablet}(\\
 &\quad \quad \text{MWT}_{XY} \circ \text{Cursor} \circ \text{CM} \circ \text{Mouse}(X, Y),
 \end{aligned}$$

```

    MWTS o MMouseButton (MMB)),
RRL o MMWinTablet(
    MWTXY o Cursor o CM o Mouse (X, Y),
    MWTS o MMouseButton (MMB)),
(CB > C and LSB > = C) →
    CAGF o ControlButton (CB) else <Off>,
(LSB > C) → CAGB o LShiftButton (LSB) > .

```

This description indicates that the Insect demo prefers Left Shift key over the Control key.