Interface Ontology: Creating a Physical World for Computer Interfaces

Pippin Barr Robert Biddle James Noble

Abstract

Interaction patterns are often used to present existing knowledge about user-interfaces in a generalised, but well-defined manner. Such patterns provide insight into the practises and assumptions of the design community. This paper presents a small number of patterns based on the concept of physical ontology. That is, the paper puts forward patterns in which computing concepts in a program are explained to the user through basic physical metaphors. The end result is an initial foray into uncovering the physical basis of the user-interface.

Introduction

Computers are highly complex devices; and no single person has been able to comprehend their entire working for many decades. With the appearance of the desktop computer, a new audience for computers was introduced: every day people with little or no computer expertise. This brought about the serious contemplation of user-interface design. Even before this, however, people who used computers have needed some means of understanding the inner workings so as to perform their work.

The innards of a computer or program are as complex as they are incomprehensible to most people. The workings of a computer are most basically represented by electrical impulses, which are impossible for humans to understand in detail. Next, the notion of binary instructions interpreted by various processors provides a more structured view, but is still inaccessible. This continues all the way to modern programming languages, which, although far more accessible than binary code, are still vastly too complicated to explain concepts to an average user.

Metaphor is an extremely popular approach to making complex system concepts more accessible. A user-interface metaphor uses some familiar object or concept to explain the more difficult underlying software. Thus, for example, a trashcan is used to explain the process and nature of file deletion to the user in most desktop systems. User-interface metaphor is recommended in design guidelines for both Apple [1] and Microsoft [13], for example. Additionally, considerable research into the practical nature of the concept has been performed, offering design processes [3, 6] and heuristics and guidelines [4, 12].

Although individual metaphors are considered useful, it is possible they will appear somewhat out of the blue without an overall structure. In order to present concepts to users *systematically* a more accessible *framework* for interaction would be valuable. Successful user-interfaces have used the concept of physical ontological metaphors to achieve this. A physical ontological metaphor explains a target concept by identifying it with a physical concept such as *object*, *surface*, or *container*. Ontology is the "science of being in general, embracing such issues as the nature of existence and the categorical structure of reality." [9, p.634] This paper specifically concerns the latter part of this definition: the *structuring* of reality, and more particularly the structuring of *physical* reality. This involves the already mentioned concepts such as objects, surfaces, containers and the like, which define the most basic concepts of human physical existence.

The paper is intended for reading by user-interface design experts who wish to understand the physical underpinnings of user-interfaces explicitly. Such readers should find interest in the

uncovering of various assumptions that are made in the use of physical metaphors, as well as find guidance on their use.

The research presented here follows the work of George Lakoff and Mark Johnson in *Metaphors We Live By* [11]. The approach to physical ontology relates to Lakoff and Johnson's discussion of (physical) ontological metaphors, which are metaphors based on the above kinds of concepts. As Lakoff and Johnson put it:

> Understanding our experiences in terms of objects and substances allows us to pick out parts of our experience and treat them as discrete entities or substances of a uniform kind. Once we can identify our experiences as entities or substances, we can refer to them, categorize them, group them, and quantify them - and, by this means, reason about them. [11, p.25]

Physical ontological metaphors utilise the basic categories of physical existence to explain other, less immediately understandable, experiences. Thus, inflation, for example, can be treated as a causal actor which is "backing us into a corner."

It will be shown in this paper that physical metaphors play a crucial part in defining the way that most modern desktop systems present and structure their underlying concepts. We discuss strictly *physical* metaphors in this paper, but other, non-physical, concepts of ontology such as *events*, *properties* and *times* are also applicable. Ontology is an area usually explored in philosophy, and has been examined in great detail there (see, for example, [8]).

Not only have physical metaphors informed past and present interface design, they clearly continue to be a source for further development. Ubiquitous computing is an attempt to bring computers into the *actual* real world physical ontology by embedding computers within it. Virtual reality is the process of creating an even more detailed physical ontology inside computers than is presently available to users. Finally, augmented reality research sits somewhere between these two extremes, perhaps juxtaposing a computer physical ontology alongside the real world one.

This paper approaches the notion of a physical ontology of the user-interface by presenting a series of patterns which explain some of the traditional applications of basic physical metaphors. The patterns progress through the more abstract to the more concrete, although all of them are ontological concepts rather than anything more specific. Beginning with a pattern describing the use of ontological metaphor in general, we proceed to the basic two: *object* and *space*. After this, we explore three extensions of these concepts: *surface*, *container*, and *actor*.

Although physical ontology is frequently used, as will be shown, the explicit statement of these patterns provides a valuable opportunity for reflection on current methods, as well as insight into the reasons for the patterns' existence. By presenting the physical metaphors explicitly in pattern form, the assumptions made by the design community at large can be examined and analysed. Finally, it must be noted that this paper presents only the basic beginning of a physical user-interface ontology, although further patterns will be pointed to.

1 PHYSICAL METAPHORS

(MAKE A WORLD TO INTERACT IN)

Intent

Make a computer system accessible to a user not necessarily familiar with it by presenting its interactive possibilities as being like interacting with things in the real world.

Problem

A computer is a machine which processes data in ways that have become incomprehensible to its users. The movement of binary codes through registers and various processors is a mystery.

> *How can a user understand what is going on in the computer so that they can achieve their tasks?*

Forces

The PHYSICAL METAPHORS pattern resolves these forces:

- Computer systems are foreign and define a world very different to the one users exist in.
- Computer systems are hard to learn.
- Computer systems are complex and abstract in nature.
- Computer systems do not necessarily operate within a consistent overall framework.
- Computer systems that are not accessible to their target users have no value.
- The physical world is a self-consistent system.
- Human beings understand how to interact with the physical world.

Solution

> *Use physical metaphors in the interface of the computer system to allow users to come to grips with the underlying concepts.*

Human beings understand their world most basically through ontological categories concepts. In particular, our experience of the world is very much a *physical* one. All humans know about the existence and basic properties of physical concepts such as *object*, *container*, and *surface*. The behaviour of these categories does not *surprise* or *confuse* us. By presenting abstract computer concepts through physical metaphors such as the above, the concepts become more accessible to ordinary users.

Example 1

The traditional command-line interface of Unix uses physical metaphors such as *objects* (files, devices), *containers* (directories) and *actors* (daemons, programs). Without these metaphors the underlying concepts would simply remain abstract collections of data and instructions which are generally incomprehensible. The command-line allows the user to "physically" interact with the system through text with commands such as "mv" (move), for example, which moves file objects between containers (see figure 1).

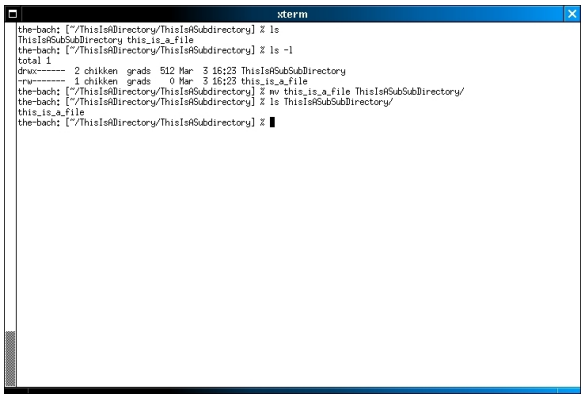

Figure 1: Physical metaphors in the command-line interface.

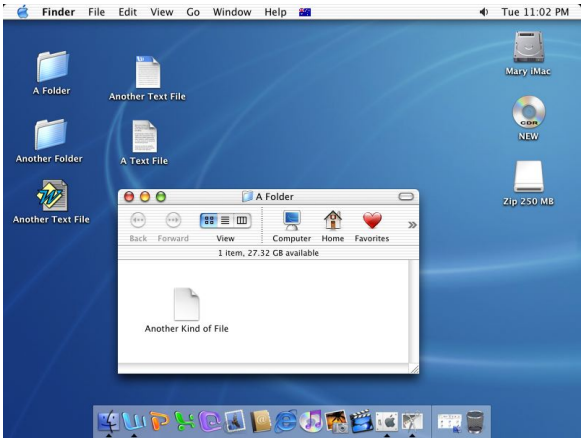

Figure 2: Physical metaphors in a GUI.

Example 2

The standard desktop GUI presents similar physical categories in a *visual*, *auditory* and *kinaesthetic* manner (see figure 2). The GUI added the abilities to *see* and directly *manipulate* the physical metaphors of the interface where the CLI provided perception and interaction only through text. Thus, users can *see* and directly manipulate containers (folders, trashcans) and objects (files, documents, pictures), and interact with actors (programs, wizards, helpers).

Consequences

+ Physical metaphors bring the computer system closer to the world the user exists in, which lowers the amount of learning necessary for basic interaction.
+ Physical metaphors entail the concept of *direct manipulation* of objects in the software [15].
+ Physical metaphors reduce the amount of abstract thought required by the user.
+ Systematic use of physical metaphors forces program concepts into a consistent overall framework.
+ Many programming languages are already written with physical metaphors in mind. Object Oriented programming languages lead to basic physical metaphors in the program code itself. Similarly, languages such as SELF [17] or Smalltalk [7] present programming concepts as objects in a physical ontology.
+ There is some suggestion that programmers *already* think of their abstract programs in terms

of physical metaphors if the claim that people think metaphorically in general is correct [11].
- Not all the abstractions of a program will fit perfectly into physical metaphors.
- It can be hard to figure out which physical metaphors to use where.
- It can be difficult to know which physical metaphors are available for use.
- It is not necessarily straightforward to translate program abstractions into physical metaphors.

See Also

- Many patterns relating to the physical categories in object oriented programming languages have already been established (see [5], for example).
- A paper by the author presents some analysis of physical ontological metaphor use in the user-interface [2].
- The Alternate Reality Kit is an explicit presentation of a physical ontology in a software system [16]. The intention is to allow novice users to "play in their own simulated worlds and to create new ones" based on "a strong analogy to the physical world." [16, p.61]
- In [16], Randall Smith also discusses the importance of "magic" in metaphor based systems. That is, it is important to allow the metaphors to deviate from their strict real-world interpretation so as to create more useful interactions than would otherwise be possible. This is a point well raised by Alan Kay also [10].
- The PHYSICAL METAPHORS(1) pattern is made possible by subsidiary patterns which implement the physical concepts necessary: OBJECT(2), SPACE(3), SURFACE(4), CONTAINER(5) and ACTOR(6), in this paper. Many more are possible, of course, as is briefly discussed at the end of this paper.

2 OBJECT METAPHOR

(OBJECTIFY THE INTERFACE)

Intent

Explain abstract computer concepts by identifying them with physical objects in the real world.

Problem

You have a some abstract concept in the computer system which needs to be explained to the user. This might be a data-structure, OO programming object or class, or any other structure or set of functions from your software. The average user is unable, or does not have time, to learn large numbers of abstract concepts when they are trying to achieve tasks.

How can you present the program concept to the user?

Forces

The OBJECT pattern resolves these forces:

- The program concept is unrelated to the user's world.
- The program concept is amorphous and invisible.
- The program concept is not necessarily encapsulated.
- The program concept cannot be interacted with in a natural way.

Solution

Use the metaphor of a physical object to present the program concept to the user.

An object metaphor presents the system concept as though it were an object from the real world. Objects from the real world have useful qualities that humans are well aware of which can be used to make system concepts less abstract and more accessible. Objects have many useful properties which are familiar to users and can be used to present abstract system concepts, such as:

Visibility Objects can be seen with the naked eye.

Visible state By extension of being visible, properties and states of the object can often be discerned simply by looking at them.

Direct manipulation Objects can be directly interacted with in a physical manner.

Physical location Objects have a location in space and tend to stay there unless manipulated as above.

Encapsulation Objects collect their properties together in one, encapsulated, place.

Differentiation Because of the encapsulation, objects can be easily distinguished from one another: they have boundaries. Additionally, object can be named because of this, giving the user the ability to refer to them.

Example

Consider the presentation of a set of data stored in the computer as a "document" (see figure 3). This uses the object metaphor to create a physical embodiment of an abstract collection of data. This, in turn, allows the user to physically manipulate the data by moving it from one place to another for example, as well as to comprehend the data as a whole.

Figure 3: A user-interface *object*: a document.

Consequences

+ The system concept can be directly seen by the user.
+ The system concept can have a visible state.
+ The system concept is quantifiable. This specifically offers the opportunity of *comparing* similar objects, for example.
+ The system concept can have a specific location, which allows the user to remember where it is and re-access it when necessary.
+ The system concept is clearly encapsulated which helps to differentiate between aspects of the underlying system and thus reduces confusion.
+ The system concept can be interacted with on a physical level such as being picked up, moved, and examined.
+ Object-oriented programming has the concept of an object built into it, making identification of potential interface objects potentially more straight-forward.
- Not all system concepts will be representable using an object metaphor.
- Most object metaphors will need further specification of their nature to be properly interacted with. For example, an "object" is hard to know how to interact with, whereas a "document" makes far more immediate sense.

See Also

- The OBJECT pattern can be made more specific for particular kinds of program concepts. See, in this paper: SURFACE(4), CONTAINER(5), and ACTOR(6).

3 SPACE METAPHOR

(SPACE: THE FINAL FRONTIER)

Intent

Provide a place for the existence of objects, as well as the concepts of depth and navigation.

Problem

Unless an object has a place to exist in, it is effectively just an abstract concept. Without a place to exist, no physical interaction is possible and objects remain only ideas in the mind.

Where can the objects be put so that their physical qualities are applicable?

Forces

The SPACE pattern resolves these forces:

- Objects need somewhere to exist in order to make their physical nature usable.

Solution

Use the metaphor of space to provide a place for objects to exist in.

A space metaphor allows the objects to be physically present in the user-interface. In fact, the existence of objects *requires* the existence of space. It enables the objects' to be visible, have locations, and be directly manipulated, for example. Additionally, space introduces the important concepts of navigation through space and also of depth, a third dimension.

Example 1

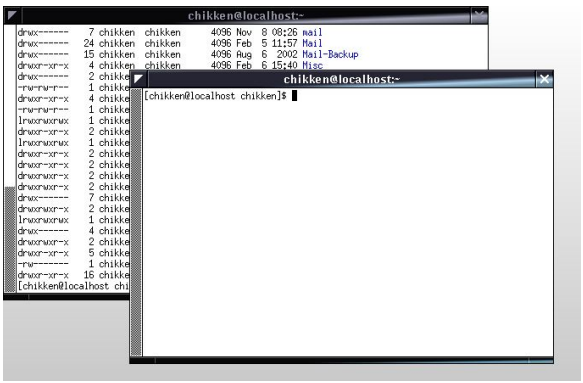

Figure 4: The use of overlap in the interface.

Space is a very general concept and thus can be found throughout the user-interfaces to software. The most obvious example is the space defined by the screen borders of a GUI. This space provides a basic area for objects to exist. It also provides the concept of *depth* which allows objects in an interface to exist at different levels such as windows which can overlap each other (see figure 4).

Example 2

The internet is often considered as a large space populated with objects called "pages." This space can be navigated through the use of hyperlinks.

Consequences

+ Gives objects an actual place to exist in.
+ Enables objects to exemplify their physical properties as discussed above.
+ Provides the concept of physical location which allows users to orient themselves in the user-interface.
+ Introduces the concept of *navigation* into the interface.
- Space can be infinite and limitless and requires bounding in some way.
- Navigation through space introduces the problem of getting *lost* in space.
- Space does not provide an immediate means for organising the objects in it.

See Also

- Jennifer Tidwell has a series of navigation-based patterns in her Common Ground collection [18]
- The SPACE pattern can be defined more specifically by creating a SURFACE(4). Additionally, a CONTAINER(5) can be thought of as specifically *enclosing* space.

4 SURFACE METAPHOR

(PUT IT DOWN)

Intent

Provide somewhere to put objects so that they can be arranged as desired and will stay in place.

Problem

Now that we have objects and space, the most basic concepts of physicality are in place. Unfortunately, most objects don't simply float in space.

Where can we put objects so that they stay where they're put?

Forces

The SURFACE pattern resolves these forces:

- Objects need somewhere solid to be put so they do not fall through space or float away etc.
- Objects need to be organisable in relation to *each other* in some way.
- Space is potentially infinite and needs to be limited.

Solution

Create a surface metaphor on which objects can be placed.

A surface provides a place for objects to be put so that they stay there. Additionally, a surface provides a means of limiting space and defining areas in space. With the ability to place objects in specific positions, the valuable function of user-imposed layout also becomes possible.

Example

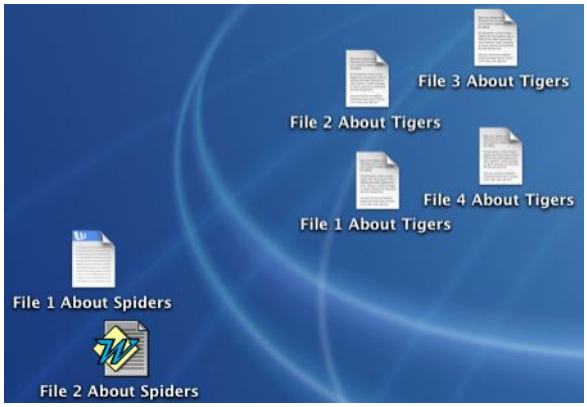

Figure 5: The use of a surface in the desktop metaphor: the desktop.

The classic example of surface is found in the Desktop metaphor of most modern operating systems. The desktop in question is a surface on which the objects in the system can rest upon and can be organised by proximity (see figure 5).

Consequences

+ A surface provides a permanent and solid location for an object.
+ A surface provides the ability to position objects in relation to each other that have meaning to the user: organisation.
+ A surface limits space such that it is no longer infinite, but is defined by the boundaries of the surface. This reduces the risk of a user being lost in space.
- A surface is a very basic and non-hierarchical place for storing objects and introduces the problem of having a *messy* surface. This is especially true when there are *many* objects.
- A surface does not completely *remove* the possibility getting lost in space.

See Also

- Jennifer Tidwell, in her Common Ground patterns, presents a pattern called *Personal Object Space* [18]. This concerns the ability to organise objects on a space, as discussed here. Additionally, Tidwell discusses more specific forms of surfaces and their organisation in four patterns: *Central Working Surface*, *Tiled Working Surfaces*, *Stack of Working Surfaces*, *Pile of Working Surfaces*.

- A CONTAINER(5) is effectively made of SURFACE(4) which enclose a well-defined SPACE(3).

- The idea of a surface and spatial organisation on a surface also fits in with concept of a "Memory Palace" [19]. This was expounded by Matteo Ricci in 1596 in "A Treatise on Mnemonics." The basic idea being to remember things by placing them in a building constructed in the mind, utilising human spatial cognition abilities.

5 CONTAINER METAPHOR

(PUT IT AWAY)

Intent

Provide a place to store objects that need to be kept in one place, organised, or remembered for accessing later.

Problem

We now have objects and a surface to put them on, but the surface can become cluttered and messy.

> *How can we organise the objects more systematically to reduce clutter and make objects more accessible later?*

Forces

The CONTAINER pattern resolves these forces:

- Beyond a certain number, objects need to be organised more systematically than being placed on a surface.
- The user doesn't want to see *all* their objects *all* the time.
- The user needs some way to act on whole *groups* of objects.

Solution

> *Create special objects called containers which can contain other objects.*

A container is a specialised OBJECT(2) made up of SURFACE(4) which enclose a SPACE(3). This allows objects on a surface to be further located *within* the space defined by containers. Thus, collections of related objects can be placed in containers to enforce a kind of organisation on the overall collection of objects. Because a container *is* an object, containers can contain other containers, which introduces the hierarchical organisation of objects.

Example

Figure 6: Container metaphors in the user-interface: folders.

The most common example of a container in today's user-interface is the use of *folders* (see figure 6. Folders contain files and other folders, which enables a hierarchical storage system. Folders in today's GUIs are largely a graphical translation of the concept of *directory* structures from older command-line systems such as Unix. In the example, the files from the SURFACE(4) example have now be organised into separate folders.

Consequences

+ Objects can be placed into relevant containers in order to remember where they are. In particular, containers allow the specific categorisation of objects.
+ Because containers are objects, containers can recursively contain other containers. This provides hierarchical storage and categorisation.
+ Hierarchical storage enhances the ability of users to store information in a way that allows them to easily locate information later.
+ Containers allow the reduction of clutter on a surface because objects can be "hidden" inside the containers: not all objects are always visible.
+ Containers allow the user to act on whole collections of objects, by manipulating the container with the objects inside it.
- Containers can potentially become full.
- The act of creating appropriate categories of containers is not an easy one. Simply having containers does not guarantee their successful use for categorisation.
- The hiding of objects in containers introduces the problem of *losing* objects.
- Navigating through containers to find things introduces the problem of getting *lost* in containers.

See Also

- Jennifer Tidwell's pattern *Hierarchical Set* discusses the means of displaying hierarchically stored information [18].

6 ACTOR METAPHOR

(MAKE THINGS HAPPEN)

Intent

Provide an explanation for why states can change in the computer system independently of the user. Additionally provide a means for the computer to directly interact with the user.

Problems

States can change in the computer system with or without the user necessarily performing an action. The computer system often needs to communicate directly with the user, rather than passively being interacted with. The user also needs to understand how it is they relate to the user-interface

> *How these state changes be explained in an accessible way?*
> *How can the computer communicate with the user in an understandable way.*
> *How can the user think of themselves in relation to the user interface?*

Forces

The pattern resolves these forces:

- The user needs to understand why states change in the interface.
- The computer system needs to communicate directly with the user.
- The user needs to have an understandable position as relates to the user-interface.

Solution

> *Use an actor metaphor to explain certain concepts in the software as well as the user's role in the system.*

An actor metaphor in the user-interface identifies some aspect of the computer system with the concept of an actor or social entity in the real world. Actors can directly act upon the world defined by the physical metaphors in the system and change their states. Most importantly, an actor has specific *intentions* which allows psychological explanations of the actor's actions. What is more, the *user* constitutes an actor in the user-interface, albeit a very important and intelligent one. The user has all the same abilities that other actors in the user-interface has, but generally more power. Although actors are a kind of object, they are not always necessarily visible to the user, and may only be perceivable when they explicitly communicate.

Example 1

The most common example of an actor in the user-interface is the program itself (see figure 7). In this example the program "RealOne Player" is notifying the user that it was unable to perform a task it was attempting via dialog box. This involves the act of communication from one actor to another.

Example 2

Microsoft's Clippy is used to socially interact with users in order to help them achieve their work (see figure 8. Although Clippy has been discontinued in the latest version of Office, Office XP, it serves as an excellent example of an actor acting in an overtly social manner: using social conventions to interact with the user.

Figure 7: An actor in the user-interface communicating with the user.

Figure 8: Microsoft's Clippy.

Example 3

Actors can also act independently of communication with the user. Thus, when Internet Explorer inexplicably deletes your "favorites" this is generally interpreted as the strange behaviour of an actor in the system. The very fact the act went unexplained and uncommunicated to the user is what makes it noticeable.

Consequences

+ Actors explain why states can change independently of the user's action.
+ Actors provide the basis for direct interaction with the user who is a actor also. In some ways this is the opposite of Schneiderman's direct manipulation, because it involves the *interface* directly acting on the *user*, rather than the other way around.
+ Actors introduce the concept of *intention* to the interface. Intention is a well-understood concept for humans, and allows psychological explanations of why things are happening.
+ Actors have the potential to interact with the user on a *social* level, which can be more comfortable. This includes the concept of *dialog*.
+ Actors introduce the concept of *roles* which provide an obvious avenue of specialisation.
- Actors can act independently of the user, and thus may not always do things the user wants.
- Actors may be foreign to the user and do not necessarily have familiar intentions or perform familiar actions.
- The introduction of social interaction also brings about the possibility of failed or problematic social situations: rudeness, miscommunication, interruption, dislike, and so forth.

See Also

- Jennifer Tidwell has a pattern *Social Space* in her Common Ground collection [18].
- In *The Media Equation* Byron Reeves and Clifford Nass specifically address the ways in which users treat computer systems like real (socially aware) people [14].

More Patterns

In this section, further patterns for extending the framework are briefly suggested:

- The SPACE METAPHOR(3) pattern entails the existence of a NAVIGATION(*not written*).
- The ACTOR(6) pattern can be made more specific with patterns such as HELPER(*not written*) or ASSISTANT(*not written*), FRIEND(*not written*), GUARDIAN(*not written*), ALLY(*not written*), LIBRARIAN(*not written*) and so on, all of which define particular social roles.
- The CONTAINER pattern can be made more specific with patterns such as ROOM(*not written*), TRASHCAN(*not written*), TOOLBOX(*not written*) and so on.
- Non-physical ontological concepts such as TIME(*not written*) and EVENT(*not written*) could also be used to extend the framework. This would be a move toward a full ontological approach to the user-interface.

References

[1] Apple Computer, Inc. Staff. *Macintosh Human Interface Guidelines*. Addison-Wesley, 1992.

[2] Pippin Barr. A taxonomy of user-interface metaphor. In Steve Jones and Masood Masodian, editors, *Proceedings SIGCHI-NZ Symposium on Computer-Human Interaction*. The New Zealand Chapter of ACM SIGCHI, 2002.

[3] John M. Carroll, Robert L. Mack, and Wendy A. Kellogg. Interface metaphors and user interface design. In M. Helander, editor, *Handbook of Human-Computer Interaction*, pages 67–85. Elsevier Science Publishers, 1988.

[4] John M. Carroll and John C. Thomas. Metaphor and the cognitive representation of computing systems. *IEEE Transactions on Systems, Man. and Cybernetics*, 12(2):107–116, March/April 1982.

[5] James O. Coplien and Douglas C. Schmidt, editors. *Pattern Languages of Program Design*. Addison-Wesley, 1995.

[6] Thomas D. Erickson. Working with interface metaphors. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design*, pages 65–73. Addison-Wesley Publishing Company, 1990.

[7] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983.

[8] Reinhardt Grossman. *The Existence of the World: An Introduction to Ontology*. Routledge, 1992.

[9] Ted Honderich, editor. *The Oxford Companion to Philosophy*. Oxford University Press, 1995.

[10] Alan Kay. User interface: A personal view. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design*, pages 191–207. Addison-Wesley Publishing Company, 1990.

[11] George Lakoff and Mark Johnson. *Metaphors We Live By*. The University of Chicago Press, 1980.

[12] Kim Halskov Madsen. A guide to metaphorical design. *Communications of the ACM*, 37(12):57–62, December 1994.

[13] Microsoft Corporation. *The Windows Interface Guidelines for Software Design: An Application Design Guide*. Microsoft Press, 1995.

[14] Byron Reeves and Clifford Nass. *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Cambridge University Press, 1996.

[15] Ben Schneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69, August 1983.

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

[17] Randall B. Smith and David Ungar. Programming as an Experience: The Inspiration for SELF. In Walter Olthoff, editor, *ECOOP '95 Conference Proceedings*. Springer-Verlag, 1995.

[18] Jennifer Tidwell. Common ground. `http://www.mit.edu/~jtidwell/interaction_patterns.html`.

[19] Frances Yates. *Art of Memory*. University of Chicago Press, 1974.