Designing Interaction

Psychology at the Human–Computer Interface

Edited by John M. Carroll
IBM Thomas J. Watson Research Center

The right of the University of Cambridge to print and sell all manner of books was granted by Henry VIII in 1534. The University has printed and published continuously since 1584.

CAMBRIDGE UNIVERSITY PRESS
Cambridge New York Port Chester Melbourne Sydney

Published by the Press Syndicate of the University of Cambridge
The Pitt Building, Trumpington Street, Cambridge CB2 1RP
40 West 20th Street, New York, NY 10011, USA
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

First published 1991

Printed in Canada

Library of Congress Cataloging-in-Publication Data

Designing interaction : psychology at the human–computer interface / edited by John M. Carroll.

p. cm. – (Cambridge series on human–computer interaction ; 4)

ISBN 0-521-40056-2. – ISBN 0-521-40921-7 (pbk.)

1. Human–computer interaction – Psychological aspects. 2. User interfaces (Computer systems) – Psychological aspects. I. Carroll, John M. (John Millar), 1950– . II. Series.
QA76.9.H85D48 1991
005.1 – dc20 90-2419
CIP

British Library Cataloguing in Publication Data
Designing interaction : psychology of the human–computer interface. – (Cambridge series on human–computer interaction, no. 4).

1. Man. Interactions with computer systems
I. Carroll, John M.
004.019

ISBN 0-521-40056-2 hardback
ISBN 0-521-40921-7 paperback

QA
76.9
.H85
D48
1991

Q1-567-669 QA2

8 Interface Problems and Interface Resources

Stephen J. Payne

Kinds of Psychology

For the sake of argument, we can consider current psychological research to belong to one of three paradigms, according to whether it is driven primarily by the exposure and scoping of phenomena, by the specification and application of general mental architectures, or by understanding the problems that people have to solve and the environmental resources that they may utilize to so do. Most experimental psychology is phenomenon-driven. A phenomenon, such as the recency effect, or semantic priming, is discovered in the laboratory (or, occasionally, noticed in everyday life and then confirmed in the laboratory), and then pursued relentlessly through a series of ever-more-intricate experiments, to discover its scope across experimental conditions. Typically, this research leads to isolated theories targeted at explaining specific phenomena. Theoretical development then progresses by the articulation of binary oppositions: Does episodic memory rely on a different system to semantic memory, or the same system? Is there one mental lexicon or two? Newell (1973), observing this pattern, complained that the game of 20 questions could never be won, as research would uncover more binary oppostions than it ever could resolve.

In place of phenomenon-driven psychology, Newell argued for the search for "models of control processes" – what today would be called cognitive architectures– which could be programmed to mimic intelligent human performance on a wide range of tasks while respecting known constraints on human information processing. His plea has been widely attended to, and several of the most influential current psychological theories are general architectures of this kind. Newell's SOAR and Anderson's ACT* are paradigm cases, and, though there are different emphases, connectionist mechanisms, such as pattern associators or the Boltzmann machine, are often described as candidate cognitive architectures.

Architecture-driven research runs into difficulties of its own. The first, and most significant is computational power. In order to capture the targeted range of empirical phenomena, architectures may have to be so flexible as to provide very few constraints on algorithms for particular tasks, and therefore to supply very little empirical muscle. The second difficulty, somewhat ironically, is that architectures themselves tend to be shaped far more by some phenomena – their "signature phenomena" (Anderson, in press) – than by others within their broad explanatory scope. So architectures never really escape from the dilemmas of phenomenon-driven research. Anderson (in press) notes that ACT* was shaped by the fan effect (Anderson, 1983), and that SOAR was shaped by the power law of practice (Newell & Rosenbloom, 1981).

None of these criticisms insists that either phenomenon-driven or architecture-driven research should be abandoned. Some phenomena are intrinsically fascinating or economically important, and worth studying in their own right, whatever the limits on their broader theoretical significance. Likewise, the problems of cognitive architectures may yet be overcome, allowing architectures that are both sufficiently powerful and sufficiently limited to support meaningful explanations, without being constrained too heavily by the need elegantly to treat one particular phenomenon. Yet the weaknesses are apparent, and there is a third alternative, which overcomes many of these weaknesses.

The third approach maintains that to understand the way the mind works, one must understand the problems it solves and the environmental resources that can be exploited in the solution of these problems. A psychology driven from an analysis of problems and resources will still need to uncover phenomena and to posit mechanisms. But the phenomena will act as clues to the construction of accounts of how people solve their problems, and the mechanisms will be offered in the service of particular solutions.

I will call this problem/resource-driven approach "ecological" as it relies so heavily on an analysis of the problems persons face in their everyday environment. In my reading, the problem/resource orientation is a key aspect of Gibson's (1979) ecological approach to perception and of Neisser's (1978) prescription for memory research; but by adopting this label for the general orientation I do not wish to embrace all the implications that these authors derive, such as the rejection of mental processing (Gibson) or the preference for phenomena over theory (Neisser). Indeed, David Marr's (1982) work on vision, with its heavy emphasis on computation, and on theory, but its disenchantment with general mechanisms, is also ecological in the sense intended here, although from his philosophy, I am rejecting the exclusive emphasis on encapsulated elementary processing modules. Marr introduced the important idea of a "computational theory": a rather awkward term to denote that analyses of problems, analyses of what has to be computed, can themselves have theoretical status, indeed preeminent theoretical status, independent of algorithms or representations. Anderson (in press) has recently applied Marr's insight to the analysis of human memory, showing that several classical phenomena may best be explained at the computational level, independent of cognitive architecture.

As a final example of the ecological approach, I would cite aspects of Newell and Simon's (1972) approach to problem solving. A cornerstone of their work is to analyze in detail the structure of the problems that are being solved. Though Marr himself is dismissive of their work, he focuses his attack on the use of production systems. (From our standpoint, another problem is the exclusive focus on artificial puzzles.) But Newell and Simon's other major integrative idea – the problem space – is, it seems to me, a candidate computational theory in Marr's sense. The work reported in this chapter will use the problem space hypothesis as a framework for ecological analysis of artifacts.

Marr (1982) offers a persuasive analogy to argue for the ecological (problem/resource-oriented) approach. To understand how a bird flies, he notes, one cannot build a theory purely from studies of feathers and wings. Instead, one needs an account of aerodynamics, of what it means to fly, and of what it takes to fly. Only in terms of the aerodynamic theory can the structure of feathers and wings make sense.

Of course, the ecological approach also has its critics. One might be worried, for example, that different problems will need different theories, so that much of what psychologists have to say will not be very general. This is one of the frequent arguments for cognitive architectures – they express the common ground of many task-dependent theories. Another approach would be to bolster the power of task analysis – to develop a taxonomy of tasks, so that the relations between tasks and theories could be structured. The chapter will offer the beginnings of a move toward such a taxonomy for interface tasks. A different kind of response to the task–theory dilemma is simply not to worry. A theory of a single task, provided it is an important enough task, would be well worth having. The tasks of human–computer interaction are, for practical purposes, of just this kind.

Kinds of Human–Computer Interaction

Psychological research in HCI has been dominated by the first two orientations; it is typically either phenomenon-driven or architecture-driven. Phenomenon-driven research in HCI, like that in psychology, is fruitful but ultimately unsatisfying. Phenomenon-driven HCI exploits "usability phenomena," close siblings of standard psychological phenomena: Strong examples include stimulus–response compatibility and the improved learnability afforded by consistent over inconsistent design. These phenomena can be directly encoded as guidelines for design and as such have had a direct and important impact on real design work. Ultimately, however, the empirical scoping of these phenomena is an inadequate basis for application of the guidelines, as discussed by Barnard (this volume).

Architecture-driven HCI research began with Card, Moran, and Newell's (1983) model information processor and has progressed through Kieras and Polson's (1985) cognitive complexity theory and Barnard's (1987) interacting cognitive subsystems to Young, Green, and Simon's (1989) Programmable User Models. The assumption underlying all this work is that there are general cognitive constraints, which can be embodied in some general architecture and which can inform the design of devices. By definition, the architectures are independent of device use. Interacting cognitive subsystems (Barnard, 1987) model the flow of information processing between separate memory stores whether the task is using a word processor or recognizing words. GOMS models (Card et al., 1983; Kieras & Polson, 1985) describe how methods for performing routine tasks may be stored and run, whether the task is editing a document or pruning a rose bush. Programmable User Models (PUMs) (Young, Green, & Simon, 1989; Barnard, this volume) rely on SOAR (Laird, Newell, & Rosenbloom, 1987), the universal architecture mentioned already.

Indeed, this is surely a strength of these models. The mind can hardly have evolved special processing modules to cope with computer systems, so one might conclude that the important cognitive psychological constraints are surely generic. I resist this conclusion. Of course there are general constraints on human information processing, but, as the ecological approach suggests, the environmental constraints of problems and resources may be more important and carry more of the explanatory load. Simon's (1969) well-known ant parable, in which he argues that the complexity of an ant's path is produced by a simple mind processing a complex world, makes exactly this point, but the usual implication drawn by scientists of the mind is that the goal of specifying an architecture may, after all, be attainable. That conclusion may or may not turn out to be fine for long-term science. In the meantime, HCI needs to import some psychology, and my own guess is that the ecological approach will be the more fruitful.

Adopting an ecological approach entails a conclusion that has profound implications for the relationship between psychology and HCI and will overlook the rest of this essay: *Thought is shaped by tools.* This proposition, in various guises, has a long history in psychology, notably in Vygotsky's (1978) "instrumental psychology," and Bruner's similar orientation (e.g., Bruner & Olson, 1977–1978), but as Norman (this volume) points out, it has not entered the mainstream of cognitive science. The ecological approach insists on such a conclusion because it is so clear that tools change the tasks people perform, and thus the basic units of analysis for ecological psychology.

Accepting the tool dependence of thought constrains realistic application strategies for the role of psychology in HCI. It entails that HCI cannot be dealt with by taking psychological findings or theories off the shelf. Instead, we must try to understand why existing devices are good or bad, and to express this understanding so that it might be generalized to new designs. This conclusion is essentially the same as that reached by Carroll, Kellogg, and Rosson (this volume) but it has been reached by a separate route. They argue from a characterization of invention as emulation; I have argued from limitations on psychological theory.

The ecological approach to HCI also dictates the *kind* of artifact analysis that must be done: One must analyze the ways in which artifacts structure people's tasks. In this volume, Norman illustrates this kind of analysis by considering a simple checklist. Computational artifacts share this effect of task restructuring that checklists illustrate, offering users dramatically different resources, and thereby posing dramatically separate problems to users, problems that simply did not exist when the artifact was not there.

More complex artifacts set up a more complex web of resources and problems. The artifacts are useful only to the extent that they offer users new resources. In so doing they set the user new problems, but, in turn, the design of the artifact may offer resources for overcoming these artifact-centered problems. For example, a text editor gives writers the ability to swiftly edit trial sentences, but only if they can work out and remember the editing commands. Modern designs provide the user with a resource for this interface problem, in the form of visible menus and simple selection protocols. To analyze an artifact, then, is to analyze this interplay of interface problems and interface resources.

A Framework for Understanding How Artifacts Restructure Tasks

In his discussion of checklists, Norman (this volume) describes their effect on tasks at two levels. First, he analyzes the specific resources offered by a checklist, and the new problems that they pose. Then he abstracts to make general claims about the way any artifact may restructure a task: It can distribute actions across time or across people, and it can demand new actions.

This discussion raises an important issue for an ecological HCI: At what level of generality should artifact analyses be conducted? I believe that rather specific analyses will prove necessary to understand the functionality of artifacts, but that some relatively general analytic tools can be developed for some issues of user interface design, because such issues are often general across a wide range of artifacts.

The work that I report here is pitched at this more general level. The framework is derived from Newell and Simon's (1972) notion of a problem space, which underlies most attempts with cognitive psychology and artificial intelligence (AI) to understand goal directed behavior. The concept of search through a problem space was developed to treat simple puzzle-solving tasks. This is no accident – the problem space treatment of such tasks is made straightforward by the clearly discrete number of states, the clear definition of goal states, the readily itemized operators, and so on. This facile analysis is usually attibuted to the puzzle being "well-structured problems," but I believe there is another reason. In all such puzzles the user can act on and perceive the problem space directly; there is no mediating artifact.

To analyze the way artifacts restructure tasks, therefore, I suggest comparing situations of artifact use with the pure problem spaces of simple puzzle solving, and itemizing the additional complexities in the user's situation, and the additional resources that are made available. It seems plausible that this top-down analysis of the space of interface issues could converge with Carroll et al.'s (this volume) bottom-up notion of user concerns. Their most general example concern, What can I do, illustrates that artifacts often make the problem space operators obscure. In my work to date, I have concentrated on three such aspects of artifact mediation, but there are no doubt further aspects that are critical. The three aspects of artifact mediation considered in this chapter are:

The device represents the user's task domain, and the user operates on these representations, rather than directly on task objects.

Operators are not effected directly; an artificial language maps operators onto actions.

Actions are tightly coordinated with the system; their availability and interpretation are sensitive to the content and timing of system responses (which may be unknown in advance).

These aspects of device use are double-edged. Each poses a particular interface problem, especially a learning problem, but each provides new information-processing resources for the user (Payne, 1990). Furthermore, the design of the artifact may itself supply resources for the user to overcome the interface problems–for example the display design, as we will see, may help the user map from operations to actions.

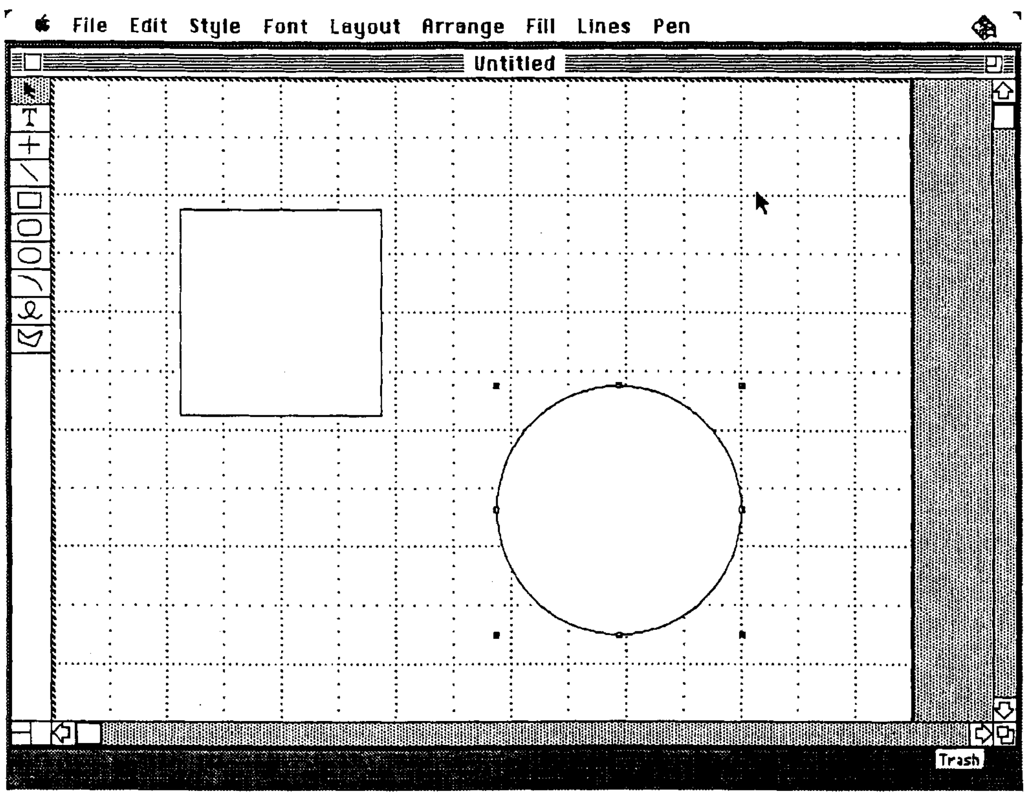

Figure 8.1. The MacDraw screen, after two objects have been created, and the circle has been selected.

The remainder of this chapter is structured as follows. The next three sections sketch attempts to understand each of these aspects and to provide simple limited models that express the understanding so that it might influence design. To provide some face validity that the models can be used to develop serious analyses of serious artifacts, and to flesh out the initial sketches, the models will then be used in conjunction to analyze a modern, popular application program.

Both to introduce the three minitheories and to illustrate their analytic potential, we will consider MacDraw, a direct manipulation drawing system that runs on the Apple Macintosh range of computers. Like many Macintosh applications, the user interface of MacDraw relies primarily on menus of tools and operations, together with direct manipulation of interface objects. Figure 8.1 shows the MacDraw display. The icons at the left are the object tools, and the menu-bar heads are pull-down menus used for file management and for editing object properties.

To introduce each of the minitheories, simple aspects of the MacDraw interface will be considered, with the emphasis on explaining the theories, through consideration of the artifact. Once each theory has been introduced, the emphasis will switch to explaining the usability of the artifact, through consideration and coordination of the theories. It should be admitted in advance that we will be considering the core of MacDraw, rather than the entire system. The central functionality is included in this core; additional interface frills are omitted in the interests of brevity.

The Device Model

Compare a display editor (such as VI) with a simple cut-and-paste editor (such as MacWrite). The display editor has separate commands for deleting words, deleting sentences, deleting paragraphs. The cut-and-paste editor allows all these to be achieved with a single command, for deleting strings. The two editors place different demands on the user's understanding. The conceptual entities (Greeno, 1983) in the appropriate problem spaces are different. Users of the cut-and-paste editor must construct the concept of a string, and the way it maps onto text objects.

This is an example of a very general interface problem for users. It reflects one of the fundamental ways in which devices restructure tasks – the conceptual objects that can be manipulated are changed. I suggest that this interface problem can be analyzed in terms of a device-oriented elaboration of Newell's problem space hypothesis, namely the yoked state space (YSS) hypothesis: The user of any device must construct and maintain at least two separate state spaces, the goal space and the device space, and a semantic mapping between them (Payne, 1987; Payne, Squibb, & Howes, 1990).

The goal space represents the "external" world that can be manipulated with the device. The minimal device space must be capable of representing all the states in the goal space. Device operators allow the user to transform states in the device space. The user's overall task is to accomplish a transformation in the goal space, but this can be achieved only by applying operators in the device space. Figure 8.2 sketches a yoked state space for the core functions of the MacDraw application. The goal space comprises a document of one or more drawings (to be more precise, line drawings with text). The user's task is to create and edit such documents. To do this, the user must construct a device space and learn how entities in this device space represent drawings and how operations in the device space can thus create and transform drawings. Figure 8.2 shows an example device space (which, as we shall see later, is rather too simple). The main conceptual entities of the device space are files and objects, where a file is defined as a set of objects in particular locations, and is in a particular state, according to whether it is saved, open, and active (there are cooccurrence restrictions here, in that only open files can be active, but these have been omitted from the description for simplicity). Objects are defined by their type, shape, size, fill pattern, and pen characteristics. A large number of operations act on these entities to create new instances and change existing ones.

The YSS is a kind of computational theory, in that it specifies a function to be computed, rather than an algorithm for its computation. It does not specify how the knowledge of goal space, device space, and so on are mentally represented; it just specifies what must be represented. The particular notation used in Figure 8.2 does

```
Goal Space

document = drawing*


Device Space

file = (object,location)*, file-state

object = obj-type,shape,size, fill-pattern, pen

obj-type = text / line / rectangle / ellipse / arc / curve / polygon

file-state = saved?, open?, active?


Operations

open new file
save file
activate file
close file
print file

create object
reshape object
move object
change pen
edit text
change object-fill pattern
rotate object
duplicate object
delete object


Semantic Mapping

document --- file
drawing --- (object, location)*
```

Figure 8.2. A yoked state space model of MacDraw.

not, therefore, have any theoretical pretensions: The intent is simply to describe the important entities and their relationships economically. A more complete psychological account would address representation and algorithm, but the computational theory as is has important empirical consequences and allows interesting analyses of user interfaces.

One way to use YSS in analysis of artifacts is to specify the "ideal" understanding a user must construct in order to exploit the device, and to examine the learnability of this specification. Payne et al. (1990) do this for the kind of cut-and-paste editor already introduced, showing experimentally that the concept of a string, and its mapping onto text objects, is indeed a central aspect of learnability. A second analytic strategy is to consider the implications of a YSS for problem solving. A particular device space and semantic mapping will facilitate some goal space tranformations and hinder others. When considering such implications of the device model, it is important to look beyond the ideal specification to alternative device models that are weaker than the ideal understanding, but that may be readily available to users and lead to problems in use. In the work on text editors, Payne et al. (1990) showed that users who fail to construct the device space concept of a copy buffer will use inefficient copying methods.

Both these analytic stategies will be pursued below, to uncover conceptual difficulties with MacDraw.

The Interface Language

The second interface problem that is common to all computational artifacts is mapping operations onto actions. A user who has constructed a yoked state space will not be able to do anything with the device before learning how to translate device operations into specific action sequences. This aspect of user interfaces can be usefully treated as a language, whether the vocabulary is made of lexical items or actions, like pointing with a mouse.

The learnability of such a language was at one time a hot topic for HCI research, but has faded from fashion. Why? Partly, I suspect, because the field has become more ambitious, wishing to impact the global design of systems, rather than mere surface aspects of the interface (the chapters in this book tend to illustrate this ambition). Partly also, perhaps, because the success of menu-driven systems has mitigated some of the critical difficulties that were apparent with earlier command-language designs (Norman, 1981) and which motivated much of the research. Nevertheless, I believe that the interface language is still a major learnability problem in many systems, and still deserving of research attention.

About eight years ago, under the direction of Thomas Green, I embarked on a project to develop a notation that can formally describe interface languages, in a way that models the structure of the language as perceived by users. The main clue driving the model was the importance of "consistency" in the learnability of interfaces. We argued that many of the important aspects of consistency can be captured by a model of user knowledge that:

Identifies the "simple tasks" that can be routinely performed and that require no iterations or branching (these correspond to the operations in the device space);

Represents these simple tasks by sets of semantic components, reflecting a categorization of the available operations;

Rewrites simple tasks into action specifications, using the machinery of grammatical rewrite rules;

Simple tasks

	Operation
Start new drawing	new
Close current drawing	close
Save current drawing	save
Print current drawing once	print

Rule schemas

```
Task [Operation] ->     action (point, "File"),
                        select [Operation]

select [Operation = new] -> action (drag, "New")

select [Operation = close] -> action (drag, "Close")

select [Operation = save] -> action (drag, "Save")

select [Operation = print] -> action (drag, "Print one")
```

Figure 8.3. A simple task–action grammar (TAG) of some MacDraw file operations. In this simple example, each simple task is defined by the value of a single feature, Operation. Rule schemas that contain unvalued features are expanded by assigning a value to the feature consistently throughout the rule. Action clauses, action (action-type, object), are the terminals of the grammar.

Marks the tokens in rewrite rules with semantic features from the task world (or from a model of semantic memory) to capture regularities in the task–action mappings.

These ideas describe a computational theory of task–action mapping. The theory can be expressed as an attribute grammar, a task–action grammar (TAG; Payne & Green, 1986). A simple example is shown in Figure 8.3, which describes the mappings onto actions of the file operations in the MacDraw device model. This example shows a key principle – structural regularity in the relationship between tasks and actions can be captured by higher level rule schemas (in this case a single higher-level schema, marked with the task symbol, suffices).

Task–action grammars illustrate the ecological approach by being constrained to illuminating a particular interface problem. Comparing their range of coverage over arbitrary user interface scenarios, with the coverage of universal architectures, as Young, Barnard, Simon, and Whittington (1989) do, is thus meaningless (as the authors recognize, but their figure 1 seems to deny). What may appear to be a weakness of TAG is, I would argue, a virtue. By narrowing their focus, task–action grammars can remain very simple and relatively concise, and yet still allow meaningful psychological insights. I hope that my subsequent MacDraw analysis makes good on this promise.

I do not mean to imply that TAG is without fault; it was initially proposed as a kind of base camp – more accurately, perhaps, a first excursion from Phyllis Reisner's (1981) base camp – and it has been gratifying that many people have taken up the assault, though on different routes (e.g., Hoppe, 1988; Reisner, 1990; Tauber, 1988). One particular weakness of TAG is relevant to the focus of this chapter: It neglects the vital role that the display can play as a resource for task–action mapping. This point is well demonstrated by the experiment of Mayes, Draper, McGregor, and Oatley (1988), who found that experienced users of MacWrite, a menu-driven user interface, could not recall the names of many frequently used menu items. This finding was replicated by Payne (1991) using an imagined task context to lessen the possibility that recall was artificially depressed by lack of goal-derived cues. Yet TAG assumes that users must learn and remember the precise mapping from operation to action, albeit in general schematic form where possible. To tackle this problem, Andrew Howes and I developed an extension to TAG – D-TAG, display-oriented task–action grammar (Howes & Payne, 1990).

D-TAG aims to analyze the way the display serves as a resource for the user in specifying operations. It uses simple formal extensions to the TAG notation, of which the most important is a display-item function, which we defined as follows:

> display-item (<task features> <display features>)
>
> retrieve the subset of objects in <current display> that have <display features>, then retrieve semantic definitions for each of these objects from <lexicon> and compare each of these definitions to <task features>. Return the best match. The function relies on two implicit parameters, a lexicon and a current display, which are not denoted in the task–action grammars.

Note that, once again, this is a (rather loose) computational theory. It specifies a function to be computed, not a particular representation and algorithm. Howes and Payne (1990) also suggest a scheme for representing display features as nested frames, each identified by some of the descriptors: "type" (e.g., icon menu), "location" (relative to the nesting frame, e.g., top) and "state" (e.g., highlighted). The key point is that display features must be capable of representing descriptions like "the item at the top of the left icon menu," as, informally, we have found that users often remember just this kind of detail.

```
Display-based rule schema

Task [Operation] ->    action (point, display-item( [Operation],
                                                    [type=menu-bar]  ) ),

                       action (drag, display-item(  [Operation],
                                                    [type=menu-bar;
                                                         type=pull-down-menu;
                                                         location=below;
                                                              type=item]  ) )
```

Figure 8.4. A display-based rule schema for the MacDraw file operations. The display-item function is used to specify the objects of action clauses indirectly, by matching display items against task semantics. The display features are described by nested frames, each separated by a semicolon.

D-TAG allows descriptions of knowledge of interface languages that rely on the available information on the computer display. For example, Figure 8.4 shows a D-TAG description for the same MacDraw file operations as we considered previously. According to the grammar, the user knows that to execute each operation you must first point to the item on the menu bar that is semantically closest to the operation, then drag down to the item on the pull-down menu that is semantically closest. According to this model, then, the screen design of MacDraw allows such a user to perform these operations without ever memorizing the specific menu items, and thus brings the TAG model of competence into line with the results of Mayes et al. (1988).

Like the YSS model, TAG/D-TAG can be used for analysis in different ways. Ideal TAGs can be used to index learnability, by specifying a part of what the competent user needs to know. More heuristically, TAGs can be used to explore alternative perceived structurings of the interface and their implications. What happens, for example, when users fail to perceive certain regularities, or perceive bogus generalizations that are ultimately misleading?

Action and Interactivity

The final type of task restructuring noted previously is coordination with the device. The balance of the issue here is more toward the resources for action provided by the user interface than toward the problems it poses.

D-TAG only treats a limited part of that issue, knowledge of the mapping of operations onto actions. Another important issue was exposed in the simple experiments already noted (Payne, 1991). Experienced users of word processors are unable to report the precise effects of a frequently used operation, such as searching for a specified string, or deleting a word. They fail to remember aspects of the dialogue that are vital for the detail of action sequences. The obvious, simple conclusion is that users do not commit to memory things that they are able to pick up from the display as required. Nevertheless this "uncertainty of effects" phenome-

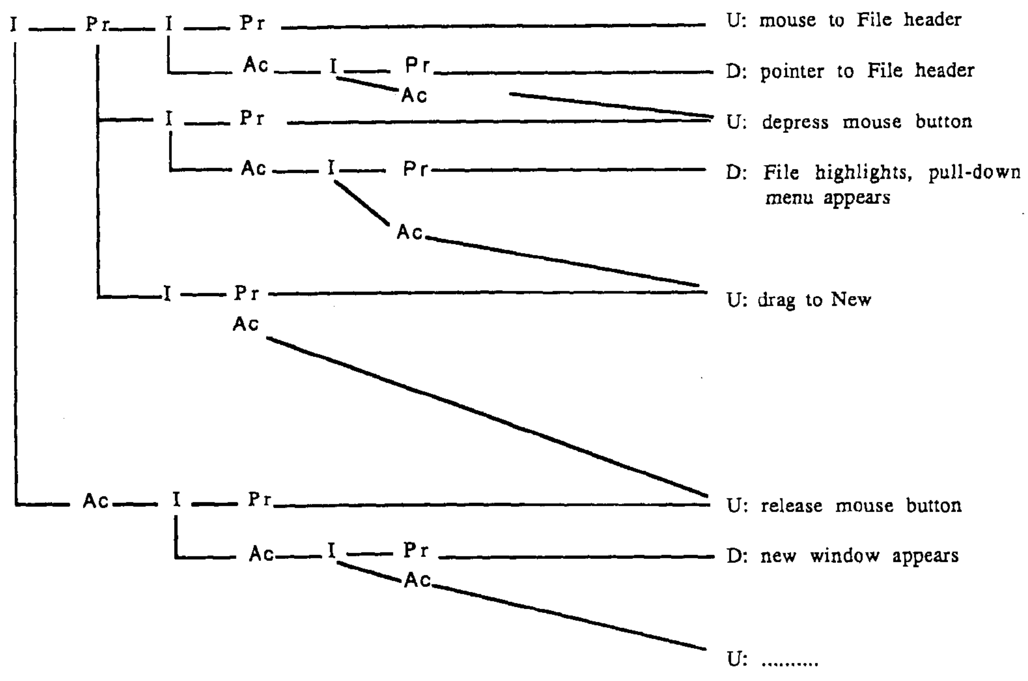

Figure 8.5. An interaction tree for opening a new file in MacDraw. U, D denote actions by the user and device respectively. Each unit interaction (I) is made up of a presentation (Pr) followed by an acceptance (Ac).

non has important implications for models of planning, suggesting a vital part of skill is the on-line interpretation of system responses, which must therefore play a much larger role than simple feedback.

I have recently begun an analysis of this feature of action at the user interface, by adapting Clark and Schaefer's (1987, 1989) model of human conversation to describe the way meaningful interactions are produced through coordinated presentations and acceptances (Payne, 1990). The key claim of this model is that interaction is structured into Unit Interactions of two phases, presentation, followed by acceptance. Both presentations and acceptances may be composed out of nested unit interactions and, in a sense, achieved collectively by user and device. What distinguishes HCI from human conversation, in this model, is that the criterion for eventual acceptance is not mutual grounding, but rather the user's sense that the interaction can be accounted for in terms of his or her current purposes. What distinguishes HCI from goal-driven action on some passive world is that this accounting process is done dynamically and incrementally, allowing the user to act meaningfully without fully articulating (at the time) goals or plans.

Figure 8.5 shows a simple interaction tree for the MacDraw new-file operation. The figure shows how the Interaction Tree model describes the interaction as a sequence and a hierarchy of unit interactions (denoted I). Each unit interaction begins with a presentation by either user (U) or device (D), and ends with an acceptance, to which both user and device may contribute, but which ultimately must be closed by the user accepting the role of the unit interaction by moving onto the next (shown by diagonal acceptance arcs). Interaction trees suggest that interaction exploits a tight coupling between user action and device responses, and suggests that many user actions primarily play a role of "accepting" previous actions.

The analytic ambitions of interaction trees are that, by displaying the conversational status of user actions and device responses, they may inform the design of these "low-level" aspects of the interface, which, I suspect, are central to the feel of the system. Unlike the TAG and YSS models, interaction trees cannot be used to describe the overall configuration of a user interface. Instead, the analyst may focus on the interaction design for particular use scenarios, or on describing actual observed interactions between a user and the machine.

An Analysis of MacDraw

The coverage of each of the three minitheories described previously is, as I keep stressing, strictly limited. This suggests that they might fruitfully be combined to provide a cumulation of insights into user interface designs. Our consideration of MacDraw file operations has already illustrated this potential, to an extent. But the most interesting usability aspects of MacDraw lie not in file handling, but in drawing. In this section, the problems and resources that are provided by the MacDraw drawing interface will be described at each level in turn.

Constructing a Device Model for MacDraw

Using MacDraw radically restructures the task of producing line drawings (compared with using pen and paper). At the highest level, the task changes from drawing to assembly – the user no longer has to draw shapes, rather, they are chosen, sized, and assembled into a complete drawing. How can we analyze this task restructuring? How can we understand its psychological consequences? The key feature of the device space for MacDraw is that it is object-centered. Graphic objects, like circles, rectangles, or text objects, can be created and modified by selecting from a repertoire of tools; drawings are simply arrangements of objects. To use MacDraw successfully, the user must appreciate the important, consistent nature of this object-based representation scheme (as is recognized by the user manuals, e.g., Claris Corporation, 1988). The user must also appreciate its limitations, which are not at first obvious, and which can cause problems to users who lack an adequate model of the device.

The simple device space for MacDraw shown in Figure 8.2 illustrates the central role played by the concept of an object. The fundamental point is that all sorts of geometric shapes are treated as objects, and that operations apply to any such objects. This allows powerful generalization, of the kind the string concept provides for text editing. Unfortunately, as we shall see, the simplicity of the conceptual model

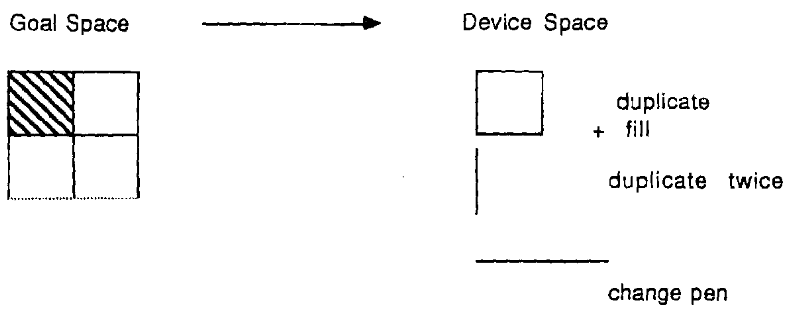

Figure 8.6. A troublesome drawing, with a possible construction trace.

shown in Figure 8.2 needs to be compromised to fully represent the device. The analysis of MacDraw usability will begin by considering the implications of the simple object-centered device model shown in Figure 8.2, which can be viewed as representing a minimal understanding required for use of the device. Next, inadequacies in this device model will be exposed, and the kind of elaborations that must be made to the model to capture the intricacies of the MacDraw design will be considered.

Mapping from the device space to the goal space takes on a rather different character with MacDraw than with text editing. In text editing, the goal space is a stable structure of relations between text objects: Paragraphs are always made out of sentences, which are always made out of words. Drawings are not so uniform: They may contain circles inside or outside squares, use unique shapes and shading patterns, and so on. It is therefore an ongoing problem for the user to map between the current goal state and MacDraw's device space, to instantiate the general semantic mapping shown in Figure 8.2.

The fact that most operations apply to any object allows powerful generalization, but it also constrains the semantic mapping, as all objects must have internally consistent attributes. Often, the user wants to specify an attribute for part of an object, but, because MacDraw knows nothing of object parts, this cannot be done.

The object-centered device model is also needed to explain a limitation on filling, which might otherwise frustrate the user. A space enclosed by a number of separate lines cannot be filled, as fill-patterns are properties of objects, not of pixel space.

These two inherent limitations of the rigid object-centered design of MacDraw both have important implications for the problem of mapping goal space drawings onto device space objects. To illustrate both issues economically with a single example, consider the task of creating the drawing shown in Figure 8.6. At first, this drawing might appear to be parsable into several different object configurations. But many of the most intuitive parses will simply not permit the drawing to be replicated. If the grid is created from horizontal and vertical lines (tempting, as the duplication operation means that each orientation would only need to be drawn once), then the cell could not be filled. If the grid is created out of four boxes, then the appearance of the bottom line cannot be changed. If the grid is parsed into a square containing a cross, then neither the cell nor the bottom line can be changed.

The only solution is to use a strange hybrid of boxes and lines. This troublesome semantic mapping is shown in Figure 8.6. Note how these problems might be compounded if the desired properties of the drawing are not all specified in advance. Certain edits of a drawing depend critically on its construction; the history of a design, as well as its final shape, influences possible future developments.

These points arise from the consistent object-centered design of MacDraw. They suggest that a user without a robust device model will often be misled. But other usability issues emerge because of quirks in the interface. The first quirky feature of the device space is that circles and squares are treated as special kinds of ellipse and rectangle respectively. These objects can be created by modifying the method for creating the more general object. The nature of these modifications is treated in the task–action grammar; the point at this level is that the user needs a particular conception of the space of geometric objects. A further quirkiness is that some objects are initially created as "frames," filled with nothing, others as "solids," filled with white. This difference is visible on the display, rather indistinctly, though it affects printed drawings only if objects overlap. It does, however, have implications for the selection of objects, as solids can be selected by pointing anywhere on their enclosed surface, whereas frames demand precise pointing to the outline.

Next, consider the grouping operation. The device space in Figure 8.2 suggests (as does the user manual) that grouping combines several objects into a single composite object, which then behaves just like any other object. However, the grouping operation in fact maintains the pregroup object structure (to implement ungrouping presumably), leading to a nasty inconsistency. If a polygon is made out of lines, and then grouped, the polygon can be moved and reshaped as if it were a bona fide object. But it cannot be filled, as the grouping operation only applies to the lines – the status of the space they surround is unchanged.

The final device space issues to be confronted concern the relationship between text objects and other "geometric" objects. In many respects, the text objects are like any others. This consistency has its upsides and its downsides. Text objects are created using a tool from the tool menu (although, obviously, the method of creation also involves typing), and they can be moved, rotated, and duplicated just like regular objects. Further, style changes must be applied to whole text objects: A single underlined word will necessitate dividing text into three separate objects–before the word, the word, and after the word.

However, text objects have some special properties, not shared by the others. First, text cannot be reshaped. If text is included in a grouped object, then that object can be reshaped, but the text will remain in the same size, and in the same position relative to the frame of the composite object. All the other objects will change relative dimensions, so that the overall effect can be unpredictable.

Second, there is a special kind of text object, called "paragraphs" in the user manual. A paragraph is created without selecting the text tool (see the discussion on D-TAG). It differs from standard text objects (called "captions" in the manual) in being fitted to the boundaries of a geometric object, if one is selected.

Third, text can be edited, using simple cut-and-paste edit functions. Editing, like creation, begins with selection of the text tool. By then pointing to an existing text

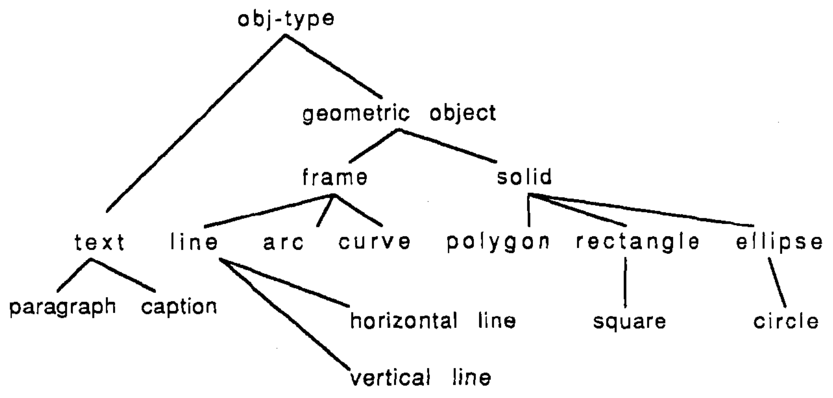

Figure 8.7. An elaborated concept of object type for the MacDraw device model.

object insertions and other edits can be accomplished. In fact, a full device model for MacDraw will contain a model for text objects similar to that described by Payne et al. (1990). However, if a text object is selected, before editing, and the text tool is not selected, the whole text object will be replaced by any new typing. Furthermore, this edit cannot be recovered.

All these properties of text objects must be understood to use MacDraw without difficulty. They can combine to produce behavior that, without the correct device model, is very hard to interpret. Imagine that one wishes to insert new text in text that is part of a grouped object. The user should realize that this cannot be done without ungrouping, as the change is not to the grouped object as a whole. However, a user with an incomplete model might fail to realize this limitation. What happens? A new text object is created, overlaying the first. This may result in the first being obscured, or, worse perhaps, if the "insert" is within a space in the original text, the failure to edit will go unnoticed until, say, the composite object is moved, and the new text object is left behind.

A very similar problem can result when the user wants to add new text immediately before or after an existing text object. Does this involve editing or creation? To choose between the two, the user must know that text objects are created with leading and trailing blanks on every line. Typing within the range thus defined will append to the existing object; typing outside these borders will create a new object.

To interact skillfully with MacDraw, then, the user needs to construct several elaborations on the device space of Figure 8.2, especially the object entity. Figure 8.7 gives a schematic of this more adequate, elaborate model of object types.

Task–Action Grammars of MacDraw

Task–action grammars of MacDraw have already been published by Payne and Green (1986), and Schiele and Green (1990). Payne and Green (1986) focused on the issue of "special" objects, circles, squares, horizontal and vertical lines, and "special" object movements, constrained to vertical and horizontal. In the MacDraw interaction language, both of these constraints are specified by holding down the shift key while performing the requisite action. However, as the TAG analysis by Payne and Green exposed, this consistency can be captured only if the shift-key depression interrupts the standard sequence of pointing and dragging. Payne and Green argue that this organizational conflict will lead many users to simply ignore (or never discover) the constrained form of movement.

A point worth noting in the current context is that Payne and Green's analysis relied on the concepts of "special" objects being available to the user. The device space and the task–action grammar are closely related. In particular, the device space specifies the granularity of simple tasks, and also provides semantic features for organizing the task–action mappings. (For an artifact analysis that rests on this chain of influence, see Payne, 1989.)

Howes and Payne (1990) use a D-TAG to analyze an inconsistency in the MacDraw interface. With all objects except text, creation of the object leads to automatic deselection of the object tool. With text, the tool remains selected until specifically deselected.

In the light of this previous work, the discussion of usability issues arising from a D-TAG analysis of MacDraw will be limited to some specific points, relating to some of those raised in the discussion of the device space. Figure 8.8 shows a partial D-TAG for the tasks of creating and editing objects. The D-TAG shows how tempting organizations of the language can produce inappropriate generalizations that will lead to errors in performance. Rules marked with an asterisk denote overgeneral schemas of this kind. Accurate performance requires knowledge of the more specialized forms marked with letters.

The first misgeneralization involves object creation. Almost all object creations begin with selection of the appropriate tool icon, which is done in a uniform way. The user might well be inclined to form a completely general creation schema (Rule 1), which cannot cope with the case of paragraph creation. The prediction is that users, especially those who do not read the manual, will rarely use paragraph text.

The second misgeneralization involves object modification. Again, almost all modifications begin with selection of the to-be-modified object. Furthermore, all "discrete" modifications, such as filling, rotating, flipping, changing font, and changing line width, can then be achieved by using a simple display-based rule (Rule 3). Unfortunately, as noted, text modification must begin with selection of the text tool, rather than a text object. Failure to encode this special case has very unpleasant results, as attempting to edit with a text object selected results in the replacement of that object. An appropriate task–action grammar therefore relies heavily on features of the more elaborate device space shown in Figure 8.7.

For the user who does construct an appropriate organization, note how closely related are the methods for editing and creating text. The organization of the task–action grammar thus compounds the potential problem noted previously, of inadvertent creation of new text objects.

Simple Tasks

	Features			
	Obj-type	Effect	Change-type	Change
Create geometric object	geometric	create		
Create text caption	caption	create		
Create text paragraph	paragraph	create		
Change object shape	geometric	modify	cont	shape
Rotate object	any	modify	discrete	rotate
Flip object	any	modify	discrete	flip
Fill object	any	modify	discrete	fill
Insert text	text	modify	cont	insert

Rule Schemas

```
*1. Task[Effect=create, Obj-type]   ->        select tool
                                              create [Obj-type]

1a.Task [Effect=create, Obj-type = geometric] ->   select tool
                                                   action (point, val-from-goal)
                                                   action (drag, val-from-goal)

1b. Task [Effect=create, Obj-type=caption] ->      select tool
                                                   action (point, val-from-goal)
                                                   action (type, val-from-goal)
1c. Task [Effect=create, Obj-type=paragraph] ->    action (point, val-from-goal)
                                                   action (type, val-from-goal)

*2. Task [Effect=modify]  ->                  select object
                                              modify [Change-type, Change]

2a. Task [Effect=modify, Obj-type=text] ->  select tool
                                            action (point, val-from-goal)
                                            action (type, val-from-goal)

2b. Task [Effect=modify, Obj-type=geometric, Change-type=discrete] ->
                                              select object
                                              modify [Change-type, Change]

3. modify [Change-type = discrete] ->

                   action (point, display-item( [Change],
                                                [type=menu-bar]  ) ),

                   action (drag, display-item(  [Change],
                                                [type=menu-bar;
                                                      type=pull-down-menu;
                                                      location=below;
                                                           type=item] ) )
```

Figure 8.8. A partial D-TAG for MacDraw.

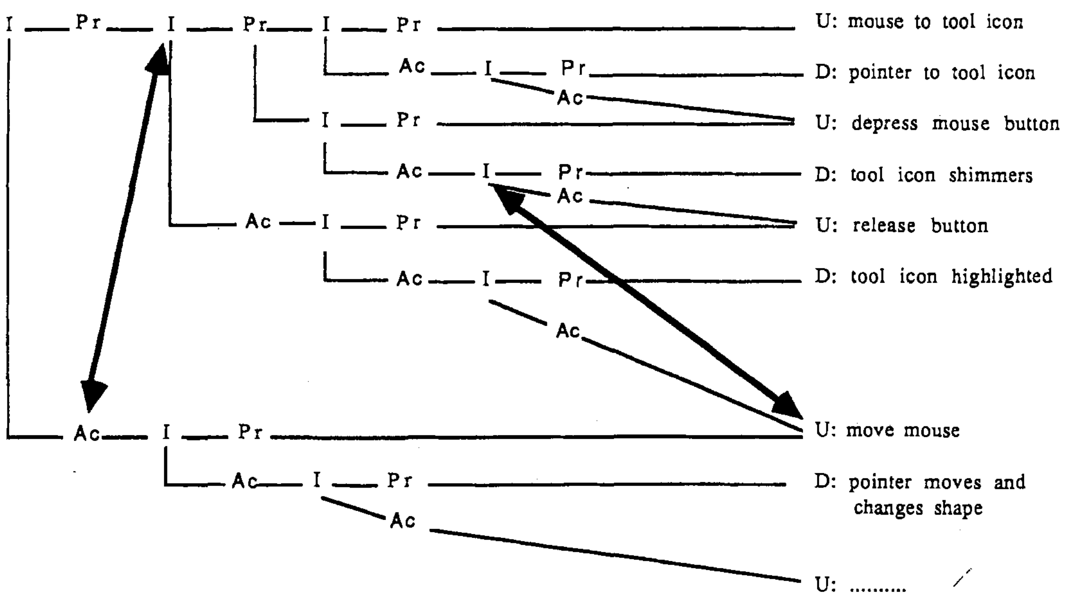

Figure 8.9. An interaction tree for MacDraw tool selection. Bold arcs show a misparsing error.

Interacting With MacDraw

The MacDraw D-TAG shows a role for the static aspects of the device's display, but it ignores the dynamics of device responses, which, I argue, play a big role in the specification of action. These aspects of interactivity can be approached with the interaction-tree notation, described previously. The interaction-tree notation is much less well developed than task–action grammars. Nevertheless, I feel that interaction-tree descriptions of MacDraw interaction patterns do throw some light on the interactivity of the interface.

Payne (1990) describes an interaction tree for MacDraw tool selection, showing how, and why, the device's response to a mouse-button depression on a tool icon can be misleading to novices. The point, briefly, is that button depression does not actually achieve anything, but MacDraw's shimmering tool icon suggests that it does, leading many novices to prematurely "accept" the selection presentation by moving (dragging) into the drawing area before releasing the mouse button to complete the tool selection. Figure 8.9 illustrates this problem with the interaction design, by showing the interaction tree for the ideal interaction sequence, and overlaying this with "parsing-error" arcs that describe a particular user's observed difficulty.

Figure 8.10 shows an interaction tree for selecting multiple objects. This method is not shown in the D-TAG. It is achieved by pointing to some place on the screen outside the perimeter of the multiple-object configuration, and dragging to "lasso" the entire configuration. In this case, the problem is that the device does not offer

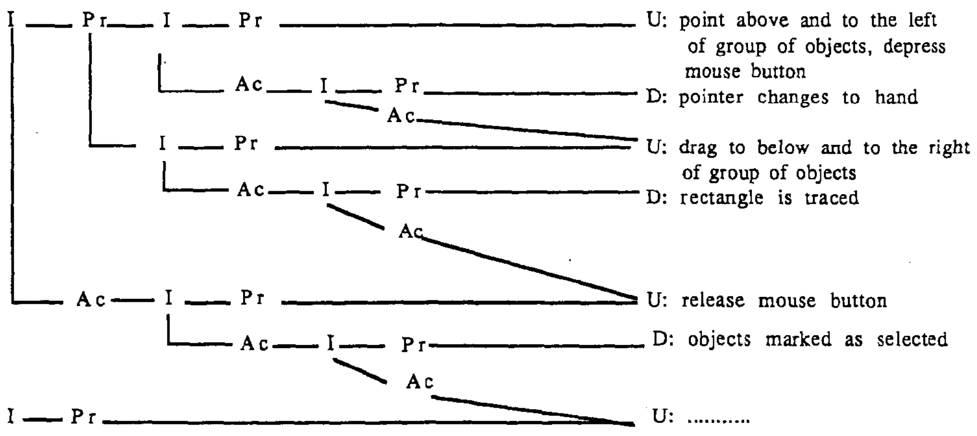
I
Pr
Ac
U: point above and to the left of group of objects, depress mouse button
D: pointer changes to hand
U: drag to below and to the right of group of objects
D: rectangle is traced
U: release mouse button
D: objects marked as selected
U: ....

Figure 8.10. An interaction tree for selecting multiple objects.

rich enough feedback to allow the user's acceptance to be informed. The interaction tree suggests that the user's action of releasing the mouse button signals acceptance of a presentation that plays the role of selecting a set of objects. Yet the device does not display which objects have been selected until after this acceptance. Ideally, the interaction should be designed so that user acceptance can be delayed until after definite feedback about the objects that have been selected.

Finally, Figure 8.11 shows the interaction trees for both editing and creating text. The point here is that the detailed behavior of the device is identical in both cases. Of course, one does not really need to draw an interaction tree to illustrate this equivalence, but one does need to focus on the low-level interactivity of the design, and no existing models support such a focus. The lack of any perceptual distinction between editing and creating text exacerbated the possibility of inadvertent creation of new text objects when the intention is to edit an existing object (or vice versa). According to our analysis, this particular interface problem is reflected at all levels, from the concepts of the device space and the organization of action sequences to the dynamics of device interactivity.

Discussion

By analyzing one artifact through the filters of three separate minitheories, I hope to have illustrated how limited models driven through the analysis of problems and resources can be integrated. Because of their limited focus, the models provide complementary insights into the psychological implications of the user interface.

The case study illustrates, I hope, that one can analyze the ways in which artifacts restructure tasks. It thus instantiates the "application strategy" that was derived earlier in the chapter from a characterization of ecological psychology. To this point in the chapter, however, I have not directly addressed a central question for this book – how might psychology impact design? I have argued that the best strategy is

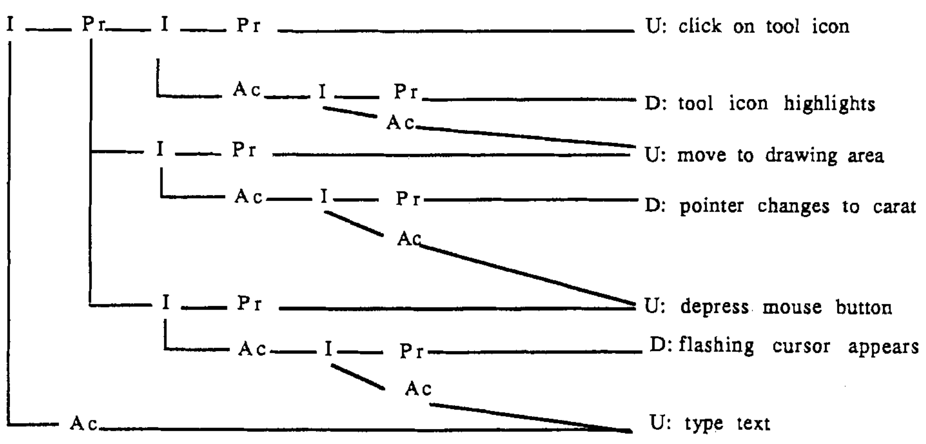

Figure 8.11. An interaction tree for editing or creating text objects.

to understand what is good or bad about existing artifacts, but I have said nothing about how such understanding might feed the design process. It is time to characterize such a vision, to make some case for the usefulness of problem/resource artifact analysis.

First, let us consider the class of design activities to which this research might be addressed. The most obvious target is redesign of existing systems. Our exposure of some potential usability difficulties with MacDraw illustrates how redesign might be driven by this kind of detailed artifact analysis. Since the analysis does not rely on empirical studies (except to test the validity of the points that arise), it can be targeted at what is often called "early evaluation," using interface specifications rather than prototypes or products. The redesign of specifications is surely cheaper than the redesign of products.

A different application focus arises from the focus of the models on analyzing the problems that users have to overcome. Such an analysis seems intuitively likely to speak to design of instructions or to other interface extensions designed to help users with these problems. In the case of MacDraw, for example, it seems plausible that instruction in the complexities of the device model could avoid many difficulties of interpretation. The user manual stresses the object model, but, to my mind, does not expose the shortcomings of this model, or the inconsistent departures from it. Users will doubtless discover some of these problems the hard way, and may be forced to build a more adequate device model through trial and error.

The experimental literature in HCI shows that instructional mental models might have beneficial effects (e.g., Bibby & Payne, 1990; Halasz & Moran, 1983; Kieras & Bovair, 1984; Payne, 1988) but nothing in that literature is able to offer specific guidelines for how to develop and convey such models. The YSS model is a constrained theory about the nature of device models. In unpublished work at the University of Lancaster, my colleagues and I have used the model to drive the design of successful instructions for a menu-driven computer system that allows remote diagnosis of faults in telephone circuits. Note that this system is removed from text editing and MacDraw, supporting the claim that the model is quite general.

Having identified a set of design activities that might be impacted, the second challenge is to package the analytical machinery so that it might be accepted into design practice (Carroll, in the introduction to this volume, calls this the applicability constraint).

The earliest work that I have described, that on task–action grammars, was driven by a simple infiltration strategy. Formalisms, like Backus Naur Form (BNF), already play an important role in some software engineering projects, for reasons that are well known and widely accepted in both software engineering and in theoretical psychology (e.g., Broadbent, 1987). It may be possible to exploit this existing niche in designers' everyday practice and value systems. TAG as a formalism is little more complicated than BNF, and the same thing is true of interaction trees, although they do not share such a well-known platform. And, though the work has not yet been done, it is easy to envisage packaging YSS descriptions in a simple uniform notation. Such simple notations may provide what Barnard (this volume) calls "application representations," packaging psychological knowledge into a tool for designers.

The status of this strategy today is, I believe, completely open. As far as I am aware, no designers outside the research community have used any of these models to analyze their designs, but this need not signal any general weakness in the idea. The particular notations could be deemed unsuitable or unusable in design contexts, or, as I strongly suspect is the case, they could be unknown. To test the strategy fully, some real effort would need to be made to sell the notations to designers. But perhaps a more direct impact on design practice could be sought by communicating to designers not the models themselves, but the design insights that the models generate. One might envisage a role for such models in the Carroll et al. scheme for supporting design emulation. The specific usability issues raised in the MacDraw analysis, for example, could be viewed as a kind of claims analysis. To use Barnard's terminology, the models might serve as a discovery representation for such claims analyses. In this role, it seems to me, these ecological minitheories offer a particular advantage. By focusing on interface problems and resources, they tend to result in analyses that cut across the individual features of an interface design. Many of the issues that emerge are caused by the configuration and interrelation of design decisions, rather than by the properties of some isolated feature. Carroll and Kellogg (1989) are aware that such interrelations are a vital contribution that artifact analysis needs to be targeted toward, but the current presentations of matrixes of claims, each tied to a concrete interface feature, tend to encourage undesired atomism.

Whichever of these two roles is envisaged for the limited models reported in this chapter (or their successors), their usefulness will depend on their ability to provoke insights into the design of artifacts that would otherwise not be readily apparent. It is a matter of fact that I had not noticed some of the usability aspects of MacDraw before undertaking the detailed analyses, and that I had not understood some of the others. To strengthen the case, further analyses of different artifacts must also succeed in uncovering and understanding aspects of usability. I fully expect such work to improve on the example models described in this paper.

Acknowledgments

MacDraw is a product of Claris Corporation. Version 1.96 was analyzed. Thanks to Jack Carroll, Robert Campbell, and Linda Tetzlaff for comments on a previous draft. Thanks to Andrew Howes and Thomas Green for discussions about MacDraw. Many of the ideas presented here were developed while the author was employed by the University of Lancaster, England. That work was supported by Alvey/SERC undergrant GR/D60355, a collaborative project with British Telecommunications plc.

References

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (in press). The place of cognitive architectures in a rational analysis. In K. Van Lehn (Ed.), *Architectures for intelligence.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Barnard, P. J. (1987). Cognitive resources and the learning of human–computer dialogs. In J. M. Carroll (Ed.), *Interfacing thought.* Cambridge, MA: MIT Press.

Bibby, P. A., & Payne, S. J. (1990). *Learning about devices by internalizing instructional descriptions* (Research Report RC 15522). Yorktown Heights, NY: IBM T. J. Watson Research Center.

Broadbent, D. E. (1987). Simple models for experimentable situations. In P. E. Morris (Ed.), *Modelling cognition.* Chichester: Wiley.

Bruner, J. S., & Olson, D. R. (1977–1978). Symbols and texts as tools of intellect. *Interchange, 8,* 1–15.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human–computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Carroll, J. M., & Kellogg, W. A. (1989). Artifact as theory-nexus: Heremenutics meets theory-based designing. In K. Bice & C. Lewis (Eds.), *Proceedings of CHI '89: Human Factors in Computing Systems.* (pp. 1–14). New York: ACM.

Claris Corporation (1988). *MacDraw user manual.* Mountain View, CA: Claris Corporation.

Clark, H. H., & Schaefer, E. F. (1987). Collaborating on contributions to conversations. *Language and Cognitive Processes, 2,* 19–41.

Clark, H. H., & Schaefer, E. F. (1989). Contributing to discourse. *Cognitive Science, 13,* 259–294.

Gibson, J. J. (1979). *The ecological approach to visual perception.* Boston: Houghton Mifflin.

Greeno, J. G. (1983). Conceptual entities. In D. Gentner & A. Stevens (Eds.), *Mental models.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Halasz, F. G., & Moran, T. P. (1983). Mental models and problem solving using a calculator. In *Proceedings of CHI '83: Human Factors in Computing Systems.* New York: ACM.

Hoppe, H. U. (1988). Task-oriented parsing – A diagnostic method to be used by adaptive systems. In E. Soloway, D. Frye, & S. B. Sheppard (Eds.), *Proceedings of CHI '88: Human Factors in Computing Systems.* New York: ACM.

Howes, A., & Payne, S. J. (1990, in press). Display-based competence: Towards user models for menu-driven interfaces. *International Journal of Man–Machine Studies.*

Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science, 8,* 255–273.

Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man–Machine Studies, 22,* 365–394.

Laird, J. E., Newell, A., & Rosenbloom, P. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence, 33,* 1–64.

Marr, D. (1982). *Vision.* San Francisco: Freeman.

Mayes, J. T., Draper, S. W., McGregor, M. A., & Oatley, K. (1988). Information flow in a user interface: The effect of experience and context on the recall of MacWrite screens. In D. M. Jones & R. Winder (Eds.), *People and computers IV.* Cambridge: Cambridge University Press.

Neisser, U. (1978). Memory: What are the important questions? In M. M. Gruneberg, P. E. Morris, & R. N. Sykes (Eds.), *Practical aspects of memory.* London: Academic Press.

Neisser, U. (1985). The role of theory in the ecological study of memory: Comment on Bruce. *Journal of Experimental Psychology: General, 114,* 272–276.

Newell, A. (1973). You can't play 20 questions with nature and win. In W. G. Chase (Ed.), *Visual information processing.* New York: Academic Press.

Newell, A., & Rosenbloom, P. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Norman, D. A. (1981). The trouble with Unix. *Datamation, 27,* 139–150.

Payne, S. J. (1987). Complex problem spaces: Modelling the knowledge needed to use interactive systems. In H. Bullinger & B. Shackel (Ed.), *Human–Computer Interaction: Interact '87.* Amsterdam: Elsevier Science Publishers.

Payne, S. J. (1988). Metaphorical instruction and the early learning of an abbreviated-command computer system. *Acta Psychologica, 69,* 207–230.

Payne, S. J. (1989). A notation for reasoning about learning. In J. Long & A. Whitefield (Eds.), *Cognitive ergonomics and human–computer interaction.* Cambridge: Cambridge University Press.

Payne, S. J. (1990). Looking HCI in the I. In D. Diaper (Ed.), *Human–Computer Interaction: Interact '90.* Amsterdam: Elsevier Science Publishers.

Payne, S. J. (1991, in press). Display-based action at the user interface. *International Journal of Man–Machine Studies.*

Payne, S. J., & Green, T. R. G. (1986). Task–action grammars: A model of the mental representation of task languages. *Human–Computer Interaction, 2,* 93–133.

Payne, S. J., Squibb, H., & Howes, A. (1990, in press). The nature of device models: The yoked state space hypothesis and some experiments with text editors. *Human–Computer Interaction.*

Schiele, F., & Green, T. R. G. (1990). HCI formalisms and cognitive psychology: The case of task–action grammar. In M. Harrison & H. Thimbleby (Eds.), *Formal methods in human–computer interaction.* Cambridge: Cambridge University Press.

Simon, H. A. (1969). *The sciences of the artificial.* Cambridge, MA: MIT Press.

Reisner, P. (1981). Formal grammar and design of an interactive system. *IEEE Transactions of Software Engineering, 5,* 229–240.

Reisner, P. (1990). What is inconsistency? In D. Diaper (Ed.), *Human–Computer Interaction: Interact '90.* Amsterdam: Elsevier Science Publishers.

Tauber, M. (1988). On mental models and the user interface. In G. C. van der Veer, T. R. G.

Green, J. M. Hoc, & D. M. Murray (Eds.), *Working with computers: Theory versus outcome*. London: Academic Press.

Vygotsky, L. (1978). *Mind in society: The development of higher mental processes*. (M. Cole, V. John-Steiner, S. Scribner, & E. Souberman, Eds.). Cambridge, MA: Harvard University Press.

Young, R. M., Barnard, P., Simon, T., & Whittington, J. (1989). How would your favourite user model cope with these scenarios? *SIGCHI Bulletin, 20(4),* 51-55.

Young, R. M., Green, T. R. G., & Simon, T. (1989). Programmable user models for predictive evaluation of interface designs. In K. Bice & C. Lewis (Eds.), *Proceedings of CHI '89: Human Factors in Computing Systems*. New York: ACM.