A Visual Language for Knowledge Acquisition, Display, and Animation by Domain Experts and Novices

COINS Technical Report 90-11
February 1990

Dirk E. Mahling

Collaborative Systems Laboratory [1]
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003
USA

[1]Telephone: (413) 545-3147; E-mail: mahling@cs.umass.edu

In Erinnerung an meine Grosseltern,

Beda Mahling und Willie Jahn.

Acknowledgments

I would like to express my gratitude to Bruce Croft for having given me the opportunity to explore issues that traditionally do not fit clearly in any established academic category. Giving me an academic home in a world where computer science and psychology are often considered to have no overlap was an unusual and kind gesture.

I would like to thank Bruce for his continuous discussion, his encouragement and enthusiasm. His guidance and pragmatic view of the issues helped me to navigate the many difficult passages in the course of this research.

Bruce Coury helped me to keep a perspective on the usability of the system. I owe to him that the real users were not buried under the theoretical models of users. Bruce's experience in experimental research into interfaces enhanced my understanding of the issues concerned in the evaluation of my system.

My discussions with David Stemple indicated how science and engineering were coming together in this research, while Victor Lesser guaranteed the relevance of my work in the area of Artificial Intelligence.

This work would not have been possible without the support of all the members of the Collaborative Systems Laboratory at the University of Massachusetts. Our daily discussions of the problems involved in knowledge acquisition, planning, cooperative work, data display, cooperative modeling etc. clearly added

to my understanding of the domain. I would like to thank Larry Lefkowitz and Carol Broverman in particular for their time and willingness to listen and discuss issues that seemed irrelevant at times. On the implementation side I owe some insights to Larry. His criticism of my programs and algorithms helped make DACRON a better system. I am also indebted to Oddmar Sandvik for assisting me in the coding of display routines for the knowledge display and advice part of the DACRON system (internally referred to as ODDCRON).

For encouraging me to expand my horizons and continue my education and research in the United States, I would like to thank my former advisor Professor Friedrich Wilkening. Without Professor Wilkening's support I would have never dared to continue my graduate student career in a foreign country.

On a financial note, I would like to extend my gratitude to the Rome Air Development Center (RADC), the Fulbright Commission, and the Germanistic Society of America who helped my studies in the United States by means of grants and scholarships.

On the private side, I have to thank my wife and my family. Sabine stood by my side through all these difficult years. She shared success and failure and more than once reminded me of the richness of life. I would also like to thank my parents, Sibyl and Ernst Mahling, for their dedicated and loving transatlantic support during my graduate student "career". Their faith in me and their belief that this dissertation would eventually come to an end helped me to persevere.

I would like to thank Brian "Hoser" Kowalski for being my hanggliding buddy all these years. I will never forget the countless hours we soared Mt. Tom and Mt. Skinner, the view of New England foliage from 4,000 feet AGL, making the

whole Connecticut River Valley look like a colored carpet with red and silver threads.

I would also like to thank my friends at Aikido of Northampton for helping me on a day to day basis to take my mind off programming problems and pay attention to their attacks and techniques, thereby allowing me to attack the problems from different angles and with new energy.

ABSTRACT

A VISUAL LANGUAGE
FOR KNOWLEDGE ACQUISITION, DISPLAY AND ANIMATION
BY DOMAIN EXPERTS AND NOVICES

FEBRUARY 1990

DIRK E. MAHLING

CAND. INF., CAND. PSYCH., DIPL. PSYCH.
TECHNISCHE UNIVERSITÄT CAROLO WILHELMINA, BRAUNSCHWEIG,
GERMANY

M.S., PH.D.
UNIVERSITY OF MASSACHUSETTS, AMHERST, USA

Directed by: Professor W. Bruce Croft

To make plan-based expert systems more accessible to end users we should develop interfaces that make the functionality of the planner seem natural and immediately understandable. In particular, we should design a graphical language and interface for the acquisition and display of plan knowledge, where the intended users are domain experts and novices. Lack of previous computer experience should not affect their performance.

On the basis of existing theories in cognitive science and from our own experimental research, we propose a model of the user's view of tasks. The model postulates the domain experts' ability to recall relevant parts of self performed tasks. The validity of the model is demonstrated in a paper-and-pencil experiment involving 153 subjects.

Employing an extended cognitive systems engineering approach we use the model, a stage process model of knowledge acquisition, and requirements from the plan formalism to specify DACRON, a visual language for plan acquisition and display embedded in a direct manipulation interface. DACRON supports the acquisition of plan knowledge by providing graphical representations of domain entities from the users' point of view. DACRON checks the consistency of specified units and graphically aids the debugging process. DACRON also allows the animated presentation of results of the planning process and its results.

To evaluate the usability of DACRON and the relevance of the acquired and displayed knowledge, experimental studies involving 39 users are conducted. The studies show that over 90% of the subjects can easily use DACRON to enter knowledge, and 80% of the entered knowledge is relevant and correct. In the case of knowledge display, subjects are able to use the displayed knowledge effortlessly and apply it to solve 95% of the problems presented to them.

Table of Contents

Acknowledgments v
Abstract viii
List of Tables xvii
List of Figures xviii

Chapters

1. Introduction 1

1.1 Declarative Knowledge and Activity Knowledge 3

1.1.1 Classes of Activity Knowledge 4

1.1.2 Capturing Knowledge 5

1.1.3 Knowledge-Based Expert Systems 7

1.2 Acquisition of Goal-Based Activity Knowledge 8

1.2.1 Direct Knowledge Acquisition Subprocesses 11

1.2.2 Acquisition of Goal-Based Activity Knowledge . . . 14

1.2.3 Interfaces for Knowledge Acquisition 17

1.3 Applying Knowledge-Based Systems 18

1.3.1 Classes of Knowledge Application 18

1.3.2 Knowledge in Context 20

1.3.3 Knowledge Display Interfaces 23

1.4 Dissertation Goals and Research Issues 24

1.5 Overview of Dissertation 26

1.5.1 Integrated Cognitive Systems Engineering Approach 27

1.5.2 Mapping the Cognitive and Functional Space 28

1.5.3 Designing the Acquisition and Display System . . . 29
1.5.4 Specifying an Architecture 30
1.5.5 Functionality 34
1.5.6 Usability Studies 38
2. Underlying Requirements, Theories and Techniques . . . 41
2.1 Knowledge Acquisition 42
2.1.1 Knowledge Acquisition Requirements for Planners . 42
2.1.2 Knowledge Acquisition Systems 47
2.1.3 Knowledge Elicitation 51
2.1.4 Knowledge Specification 54
2.2 Applying and Displaying Knowledge 57
2.2.1 Graphical Representation of Data Structures 58
2.2.2 Animation and Graphical Simulation 59
2.2.3 Office Automation and Office Information Systems . 61
2.2.4 Plan Based Assistance 64
2.3 Human Computer Interaction 68
2.3.1 Cognitive Theories of Activity 68
2.3.2 Cognitive Engineering 77
2.3.3 Interface Design Guidelines 79
2.3.4 Interface Design Philosophies 83
2.3.5 Usability Engineering and Evaluation 85
2.4 Summary of Underlying Requirements, Theories and Techniques 87
3. Relating the Human View of Tasks to Plan Formalisms 89
3.1 A Framework for the Recall of Subject-Performed Tasks . 91
3.1.1 Operationalization of Enactment 94

3.1.2 Set of Hypotheses 98
3.2 Pilot Study 99
3.2.1 Existence of Enactments 100
3.2.2 Decomposability of Enactments 102
3.2.3 Difference between Post-Situation and Operations . 103
3.2.4 Goal and Pre-Situation Determine Recall 104
3.2.5 Evaluation of the Results of the Experiments 105
3.3 Enactment Theory Experiments 106
3.3.1 Goal and Pre-Situation determine Recall of Enactments 107
3.3.2 Sequencing and Decomposing Enactments 112
3.3.3 Unanticipated Effects of Operations 114
3.3.4 Distinguishing Post-Situations from Operations . . . 115
3.3.5 Comprehensive Results 116
3.4 Post-Situation Reexamined 117
3.5 Summary of Enactment Theory 120
4. Implications for Plan Specification and Display Systems 122
4.1 Architectural Implications 124
4.2 Functional Implications 127
4.3 Interface Implications 128
4.4 Review of Proactive Cognitive Engineering Approach . . . 130
4.5 Summary of Implications 131
5. DACRON Architecture 133
5.1 Underlying Knowledge Base 136
5.1.1 Objects 139
5.1.2 Activities 140

5.1.3 Scenarios 140
5.2 Icons 144
5.2.1 Proto-Icons 145
5.2.2 Instance Icons 145
5.2.3 Valueclass Icons 145
5.2.4 Node Icons 146
5.3 Editors 147
5.3.1 Activity Editor 147
5.3.2 Object Editor 151
5.3.3 Scenario Editor 153
5.4 Consistency Checker 155
5.5 User Interaction Monitor and Suggestions 157
5.6 Knowledge Base Navigation Facility 158
5.6.1 Icon Configuration in Knowledge Base 158
5.6.2 Locating Icons in the Knowledge Base 159
5.7 Clipboard 160
5.8 Advice Facility 160
5.8.1 Plan-Networks 161
5.8.1.1 Goal Nodes 162
5.8.1.2 Agent Nodes 162
5.8.1.3 Tool Nodes 163
5.8.2 Graphical Plan-Networks 163
5.8.3 Advice Control Panel 164
5.9 Architecture Summary 165
6. USER-DACRON INTERACTION 167
6.1 Browsing 168

6.1.1 Scrolling 168
6.1.2 Reordering 171
6.2 Searching 173
6.2.1 Searching Alphabetically 173
6.2.2 Searching for Objects 174
6.2.3 Searching for Activities 175
6.2.4 Searching for Scenarios 176
6.3 Viewing 177
6.3.1 Viewing Objects 177
6.3.2 Viewing Activities 178
6.3.3 Viewing Scenarios 179
6.4 Specifying Units in the Knowledge Base 180
6.4.1 Iconic Object Specification 182
6.4.1.1 Creating New Classes 183
6.4.1.2 Creating Instances of Classes 184
6.4.2 Iconic Activity Specification 185
6.4.2.1 Iconic Specification of Wff's 185
6.4.2.2 Variables in Iconic Wff's 191
6.4.2.3 Specification of Decomposition 193
6.4.2.4 Graphical Specification of Plan Rationale . 195
6.4.2.5 Specification of Effects 197
6.4.2.6 Consistency Checking and Additional Elicitation 198
6.4.2.7 Creation by Modification and Updating . . 199
6.4.3 Iconic Scenario Specification 200
6.4.3.1 Modifying and Updating Existing Scenarios 202
6.4.3.2 Merging Scenarios 202
6.4.4 Deleting Icons and Wff's 203
6.5 Soliciting Advice 205

6.5.1 Initiating an Advice Session 206
6.5.1.1 Iconically Binding Variables 207
6.5.1.2 Iconically Selecting a Scenario 208
6.5.2 Animating the Planning Process 208
6.5.3 Setting Display Options 212
6.6 Summary of the Functionality of DACRON 213
7. Evaluation and Usability 215
7.1 Deriving Usability Criteria 216
7.1.1 Usability Criteria for Knowledge Acquisition 218
7.1.2 Usability Criteria for Knowledge Display and Advice 221
7.1.3 Usability Criteria for Interface Interaction 223
7.2 Usability Study 1: Knowledge Acquisition in the Laboratory 225
7.2.1 Experimental Setting for Knowledge Specification . 225
7.2.2 Knowledge Specification Results 226
7.3 Usability Study 2: Knowledge Acquisition in the Field . . 227
7.3.1 Experimental Setting for Knowledge Elicitation . . 228
7.3.2 Knowledge Elicitation Results 229
7.4 Usability Study 3: Displaying Advice 229
7.4.1 Experimental Setting for the Display of Advice . . . 230
7.4.2 Advice Display Results 231
7.5 Usability Study 4: Manipulation of Interface Entities . . . 232
7.6 Parameters concerning DACRON's Performance 234
7.7 Evaluation Summary 235
8. Conclusions and Future Research 237
8.1 Review of Results 237

8.2 Future Research Directions 238

8.2.1 Supporting all Stages of Knowledge Acquisition . . 239

8.2.2 Formal Methods for Cognitive Systems Engineering 242

APPENDICES

A. IMPLEMENTATION NOTES 245

B. INSTRUCTIONS FOR SPECIFICATION EXPERIMENT 255

BIBLIOGRAPHY 257

List of Tables

1. Dimensions of Expert System Use. 19

2. Repertory Grid Summarizing Knowledge Acquisition Tools. 48

3. Precondition/Goal Experiments Summary. 110

4. Planned and Observed User Performance for Knowledge Specification. 228

5. Planned and Observed User Performance for Knowledge Elicitation. . 230

6. Planned and Observed User Performance in Display Experiment. . . 232

7. Planned and Observed User Performance for Micro Goals. 233

List of Figures

1. Knowledge Distribution in a Domain. 6
2. Domain Independent Knowledge-Based Expert System. 7
3. Traditional Knowledge Acquisition Process. 9
4. Direct Knowledge Acquisition and Display. 10
5. Knowledge Acquisition Processes. 15
6. Research Goals, Issues and Related Fields. 25
7. Activity Editor and Object Editor. 31
8. DACRON/POLYMER Block Diagram. 32
9. Windows representing Instances, Classes and Valueclasses. 33
10. Specifying an Activity. 36
11. Graphical Plan Networks. 39
12. Purchase Task expressed in POISE Formalism. 63
13. Building a House in POLYMER Plan Formalism. 67
14. Operationalization of Enactment. 96
15. State-Action Graph for "checking-out-a-book" 107
16. DACRON Architecture. 137
17. Predefined Bitmaps and Generic Labels. 138
18. Instance and Class for Objects. 141

19. Activity in Goal-Based Formalism. 142
20. Activity Hierarchy in the Home Construction Domain. 143
21. Scenario for Rural House Building. 144
22. Valueclass Icons. 146
23. Activity Editor from House Construction Domain. 149
24. Hierarchy of Holders and Icons in Iconic WFF. 150
25. Constraint as Iconic WFF. 151
26. Object Editor. 152
27. Scenario Editor from the Construction Domain. 154
28. Advice Control Panel. 164
29. Knowledge Base Windows. 169
30. Flying over Knowledge Base. 170
31. Menu on Knowledge Base Windows. 171
32. Knowledge Base ordered Alphabetically. 172
33. Knowledge Base ordered by Bitmap. 172
34. Menu on Icon for Empty Form. 182
35. Pop-up Menu: Type or Copy to Set Value. 184
36. New Editor with Subject Icon. 187
37. Choosing the Top Predicate. 188
38. Completely specified Wff. 189
39. A complex iconic Wff. 190
40. Labeling a Variable. 192

41. Copying existing Icon for Subgoal. 194
42. Specifying Subgoal Wff. 196
43. Plan Rationale Specification. 197
44. Merged Scenarios. 204
45. Selecting Task for Advice and Binding Variables 207
46. Choice of Situations for Activity. 209
47. Graphical Plan Network with Highlighted Node. 210
48. Refined Plan Network with Editor for Activity. 211
49. Architecture of Adaptable Interface. 244
50. Graphical Hierarchy of DACRON Editors. 246
51. Graphical Hierarchy of DACRON Icons. 247

CHAPTER 1

INTRODUCTION

Ach, die Welt ist so geräumig,
und der Kopf ist so beschränkt.
(Oh, the world it is so spacious,
and the head is so confined.)
— WILHELM BUSCH
(Hanoverian Cartoonist, 1832-1908)

Knowledge is increasingly becoming a valuable resource in our society, in particular knowledge concerning our pursuit of complex activities. Managing a global ecology and economy, designing large scale technological systems or supervising major construction projects to name just a few, are fields that require vast amounts of knowledge in many different areas. It is therefore of growing importance to explicitly manage knowledge in order to help individuals and organizations cope successfully with complex domains. Computers can help manage this knowledge and aid our endeavors.

One of the hardest problems in applying computers to the task of managing knowledge lies in the quick and reliable acquisition of the knowledge in a form

suitable for a computer. Knowledge concerning the conduct of sophisticated activities can only be acquired from experts in a domain. Equally important is the appropriate use of that knowledge to benefit individuals and organizations in pursuing their objectives. The root of both problems lies in the question of compatibility between human and computer views of knowledge.

In this dissertation we extend the cognitive systems engineering approach to deal with interface and system design based on cognitive theories, user participatory design techniques, rapid prototyping and usability engineering. This new, integrated approach to the design of cognitive systems consists of the following steps:

1. analysis of the cognitive reality of task domain and human;
2. construction of a problem-specific theory;
3. implications from the theory for system and interface design;
4. translating design implications to design specifications;
5. rapid prototyping of the system and user participatory design;
6. usability testing of the whole system.

Orthodox cognitive engineering is augmented by the active use of cognitive theories for the design of systems. Existing design and evaluation techniques are integrated in a coherent methodology.

Our research tries to deal with what is known as "the knowledge acquisition bottleneck" in an efficient way by reducing the importance of knowledge engineers as mediators between experts and computers. We propose a system that enables experts to communicate directly with the computer. In particular, we focus on computer systems that employ a goal-based representation for

activity knowledge. Goal-based views of activities take the teleological nature of higher level human behavior into account. This representational paradigm allows capturing sophisticated activities in application domains and seems to relate naturally to the human view of the same activities.

We also present a method for using the acquired knowledge to support individuals in their professional pursuits. Since we are dealing with individuals that are interested in the domain and not in computers or knowledge per se, we have to build acquisition and display systems that are easy to understand and seem natural to their users.

1.1 Declarative Knowledge and Activity Knowledge

The knowledge necessary to perform a job falls into a number of categories. We have already mentioned skills, knowledge about procedures and knowledge about relationships and facts in a certain field. We see that knowledge is more than a mere collection of information. It is highly structured and enables individuals and the organizations to act in an intelligent way.

We can distinguish knowledge pertaining to the facts and objects in a domain from knowledge that is concerned with the conduct of an activity. The first category of knowledge is called declarative knowledge, for example "U.S. appliances run on 110 Volts", while the second category is usually called procedural knowledge [And83], e.g. knowing how to build a house. We prefer the term *activity knowledge* for the second category, because it does not suggest a particular representational formalism.

1.1.1 *Classes of Activity Knowledge*

Activity knowledge describes a large number of diverse activities. Protecting one's eyes, flipping a switch, controlling a machine, or designing a home are all activities, but obviously not on the same level.

One way to categorize activities is based on the internal, human control of activities [Ras86b,Ras86a,FP62,Whi27]. On the lowest level we find skill-based activities that take place without conscious control. These activities are smooth, highly automated and integrated patterns of behavior. Tracing lines or simple assembly tasks fall in this category.

In the next category, rule-based behavior, we find the conscious control of a sequence of skill-based behaviors. The control is stored in form of rules or procedures that can be acquired by trial and error, observation, communication, etc. Rules are matched to situations that contain elements fitting the rule.

In unfamiliar or slightly different situations, rules cannot be applied. The control of performance must move to the next category of activities, goal-based behavior. In this category, the goal is explicitly formulated, based on an analysis of the situation and the general intentions of the person. Rule and skill-based behavior is subsumed by goal-based behavior. Different behaviors are tested mentally against the situation and the goal. At this level of functional reasoning, the person constructs a model of the situation and of the possible activities.

Goal-based activity knowledge represents the most sophisticated form of activity knowledge in this categorization. It allows the application of existing skills and rules in a constantly changing environment. More and more job positions require an increasing amount of goal-based activity knowledge. The know-how

of an organization can be viewed as the combined goal-based activity knowledge of its members.

1.1.2 Capturing Knowledge

The value of goal-based activity knowledge in our modern worlds suggests we ought to make it more tangible. Giving knowledge an existence in its own right creates a number of advantages. It allows us to transfer knowledge easily from one person to another. We could make this transfer process very specific by selecting only appropriate parts of the knowledge. It also allows the transfer of knowledge to locations where highly skilled individuals are not available or to environments hazardous for humans.

Compiling bodies of activity knowledge helps to relate previously unconnected areas of knowledge. This allows us to tackle old problems in new ways and to cope with a whole set of previously intractable problems. Connecting and inspecting activity knowledge enables us to detect contradictions and ambiguities. Using the explicit representations of activities we can help individuals explore new domains or develop different scenarios to solve problems. We can also aid human operators in the execution of complex tasks by sharing not only routine tasks, but also the cognitive workload.

Traditionally, knowledge was captured in textual descriptions or movie recordings. The monolithic structure and computational intractability of these paradigms prevented us from taking advantage of the described knowledge. — Advances in artificial intelligence and knowledge engineering allow us to build computer systems that can explicitly represent knowledge. These systems, called

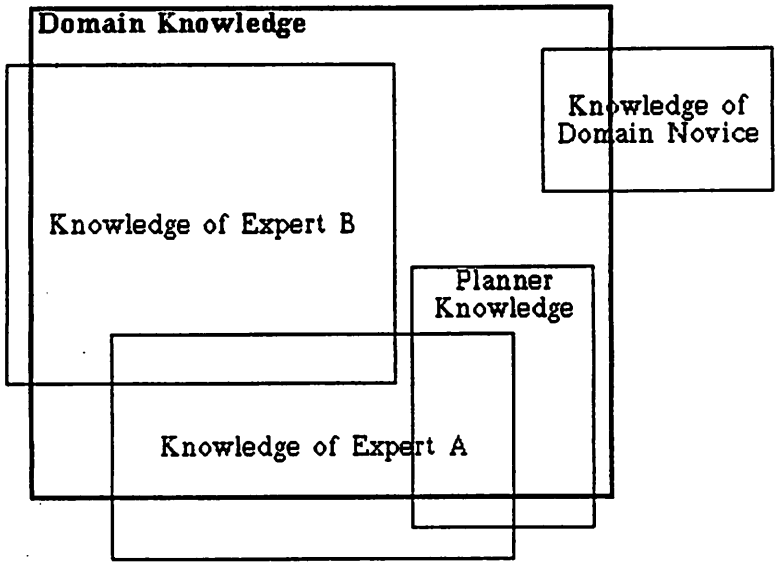


Figure 1: Knowledge Distribution in a Domain.

knowledge-based expert systems, contain a partial representation of knowledge in a domain and enable us to employ this knowledge computationally.

Figure 1 shows a possible distribution of knowledge in a domain. The large box represents the body of knowledge that defines the domain. Two experts, A and B, possess knowledge in different sections of the domain that overlaps only to a limited extent. The experts also have also knowledge outside of the domain.

A knowledge-based system captures yet another section of the domain knowledge. The knowledge of the system overlaps with that of expert A. Making the domain knowledge of experts A and B available to the system would greatly enhance its power. It could then be used to help the novice's knowledge in the domain, or to perform new tasks. The system could also assist experts by making available and using knowledge they did not previously command.

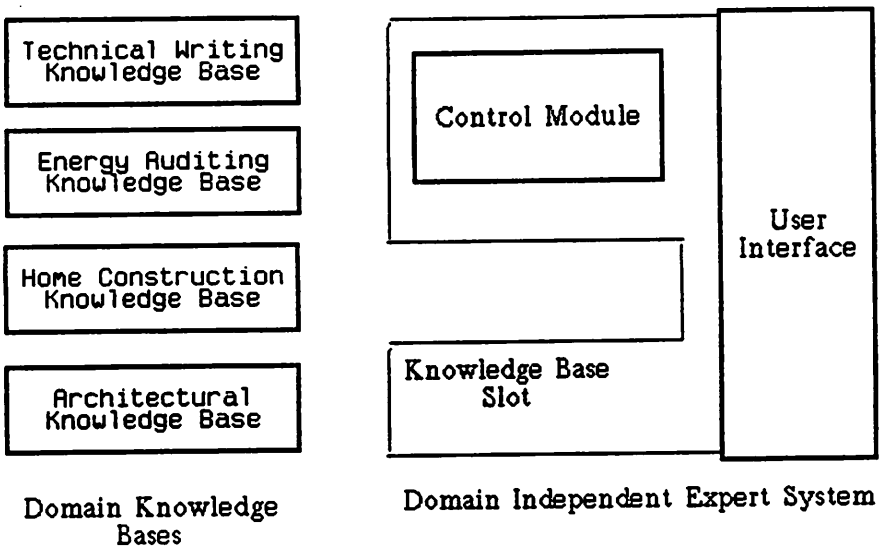


Figure 2: Domain Independent Knowledge-Based Expert System.

1.1.3 Knowledge-Based Expert Systems

The knowledge of an expert system resides in a domain specific knowledge base. A reasoning engine uses this domain specific knowledge to reason and solve problems. Separating the domain specific knowledge from the control module makes the knowledge-based system very economical by allowing quick adaptation to new domains. Instead of rebuilding the whole system, only a new domain specific knowledge base needs to be inserted.

Figure 2 shows separate knowledge bases that contain knowledge about the domains of technical writing, energy auditing and home construction. Any one of these knowledge bases could be "plugged" into the knowledge base slot in the domain independent expert system, creating an expert system for the particular domain.

The expert system contains a control module that is independent of the domain. The control module can be an inference engine for rule-based expert system [Nil80,Ste81] or a planner in plan based expert systems [CL87,HL87,Sac75]. The reasoning and problem solving power of an expert system is implemented in the heuristics used by its control module. The competence of an expert system arises from the quality and the quantity of its domain dependent knowledge. This fact makes the acquisition of knowledge an important issue.

Since we are mostly concerned with goal-based activity knowledge we will focus in this dissertation on expert systems that use goal-based knowledge representations. *Plan based expert systems*, or *planners* for short, employ goal-based descriptions of activity knowledge in their knowledge bases. In the field of planning these goal-based activity descriptions are called plans. Explicit reasoning about goals and situations is a specific characteristic of planners. Acquisition and use of goal-based knowledge (plans) becomes the critical task.

1.2 Acquisition of Goal-Based Activity Knowledge

The process of making new areas in the body of domain knowledge available to a knowledge-based system is referred to as knowledge acquisition. Knowledge acquisition is a technically challenging and time consuming undertaking.

The most common method for knowledge acquisition relies on a knowledge engineer interviewing domain experts (figure 3). The knowledge engineer will then build a prototype system and review its performance with the experts. The system usually undergoes a number of revisions before an acceptable design has

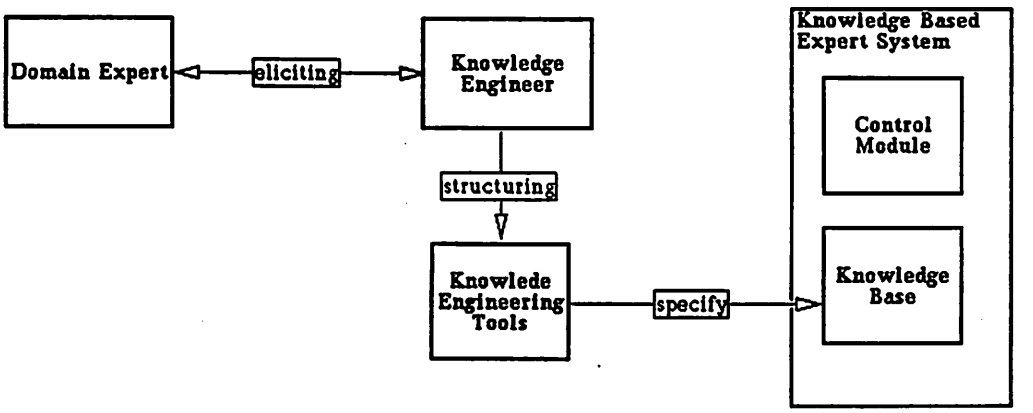


Figure 3: Traditional Knowledge Acquisition Process.

been achieved. Knowledge engineer and experts will then cooperate to fill out the knowledge base, thereby extending the amount of domain knowledge of the system. This knowledge engineering process usually takes several person-years. Change in the domain knowledge requires the renewed efforts of the knowledge engineer, even for small changes.

Knowledge acquisition has been recognized as a bottleneck in the process of building knowledge-based expert systems [Fei84]. The problems arise from a mismatch in the communication between knowledge engineers and domain experts, the sparse availability of knowledge engineers, the need for large financial expenditure, temporal efforts, and the difficulties for individuals other than knowledge engineers to integrate later changes in the domain into the system.

In this dissertation we suggest making knowledge engineering an activity that can be handled by domain experts using computers. Just as telephone users today have become their own operators, even for international calls, by using an

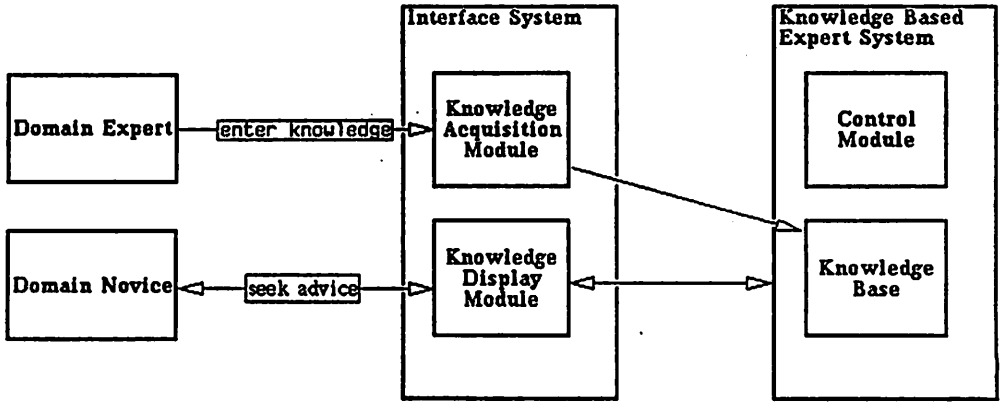


Figure 4: Direct Knowledge Acquisition and Display.

interface that makes the former telephone operators' work easily executable (the rotary dial, or the keypad), domain experts can become their own knowledge engineers when given appropriate tools.

In this dissertation we suggest the automation of the knowledge acquisition process by providing the domain experts with a visual language and a direct manipulation interface that would allow experts to enter domain knowledge directly. Direct knowledge acquisition eliminates the interview and knowledge specification process by the knowledge engineer. Figure 4 shows an integrated system for the acquisition and display of knowledge. The system can be used as a domain independent interface for knowledge-based expert systems.

By giving the domain experts such a tool we hope to eliminate the communication problem between knowledge engineer and experts. Knowledge engineers can spend their efforts on other parts of the system, improving the efficiency of system building. Domain experts using a direct knowledge acquisition tool can

update the knowledge based system when changes in the organization or the environment occur. Knowledge is acquired more quickly because many experts can enter their knowledge at the same time. Knowledge-based systems therefore become more widely available through lower building costs and shorter time investment. Running knowledge-based systems becomes more efficient, because members of an organization can augment and change the system themselves and immediately understand the advice offered by the system.

We also see tools for direct knowledge acquisition as tools for novice knowledge engineers or as means for an improved communication between knowledge engineer and domain expert, when necessary.

1.2.1 Direct Knowledge Acquisition Subprocesses

A number of subprocesses can be identified in the knowledge acquisition process. Drawing on models describing knowledge acquisition by knowledge engineers [BBB+83] and programming in general [PK83], a process model for direct knowledge acquisition can be developed [MC88].

In this model, direct knowledge acquisition from domain experts is viewed as a multi-stage process. The first stage consists of the *elicitation* of the knowledge itself. The piece of knowledge that is to be acquired by the system must be elicited from the expert. This requires at least parts of that knowledge to be present in the experts's consciousness.

Consider, for example, the domain of constructing houses. To acquire the knowledge of how houses are built, this knowledge has to be present in the mind of the expert. The expert has to understand that the general task of house

construction is involved. The expert recalls the different ways houses can be constructed, the problems associated with it, the tasks and people involved, previous incidents, etc.

In the second stage the elicited piece of knowledge must be transferred by the expert from his consciousness to the system. This stage is called *knowledge specification.* The pieces that comprise the elicited knowledge must be named and their relations must be defined. The knowledge acquisition system must provide a facility to allow this. After the specification stage is completed, the elicited knowledge is transferred to the system. A first approximation of the knowledge is represented in a formal way in the system.

In our example, the expert would have to use a certain language or medium to shape the knowledge that houses are built by erecting a frame or a shell, installing electrical wiring, installing pipes for water, etc. This specification process can result, depending on the implementation of the system, in a collection of terms like:

```
(BUILD.A.HOUSE = (first (ERECT.FRAME)
                 then  (INSTALL.WIRES, INSTALL.PIPES)
                 last  (LANDSCAPING)))
```

Verifying the semantic and syntactic consistency of the new piece of knowledge locally is done in the third stage. The new unit is regarded in isolation. Internal semantic inconsistencies and ambiguities can be detected. Missing parts and violations of the representational formalism can be found by a syntactic analysis.

If any local inconsistencies, ambiguities or other violations are found in the third stage, the person can be asked in the fourth stage to correct them. This is an iterative process, terminating with the completion of a locally correct new unit.

In the fifth stage the locally correct new unit has to be added to the existing knowledge base. This process is known as *knowledge assimilation.* The best place within the existing knowledge base for the new unit has to be found.

Adding a piece of activity knowledge, BUILD.A.HOUSE for example, could be done by searching for the closest matching unit in an object oriented knowledge base. This unit could be the CONSTRUCTION.ACTIVITIES unit. The new unit would then be made a subclass of the existing unit.

Contradictions and other sources of incongruency with existing knowledge have to be detected. That is done in the sixth stage, *global verification.* If no inconsistencies are found, the new unit is tentatively added to the knowledge base.

Contradictions in our example could arise in the sixth stage if a different BUILD.A.HOUSE unit already exists. Other incongruencies could be introduced by the violation of protection intervals demanded by an existing unit. The object, SHEETROCK for instance, might already exist in the knowledge base. BUILD.A.HOUSE might use SHEETROCK in conjunction with COPPER.WIRE, but if SHEETROCK can only be used with HIGH.VOLTAGE.WIRE in this domain, there is a clear and detectable violation.

The seventh stage applies only if unresolveable inconsistencies were found in the sixth stage. In that case inconsistencies must be resolved either by querying the user or by automatic means.

To guarantee semantic and pragmatic consistency of the updated knowledge base, an eighth stage has to be executed in the knowledge acquisition process. The behavior of the updated knowledge base, employing the tentatively added piece of knowledge, must be presented to the domain expert. This enables the expert to recognize dynamic relations between existing and new pieces of knowledge and to verify that these interactions were intended.

Running the expert system for the BUILD.A.HOUSE activity, CONCRETE might be used as building material for the frame. Suppose this was not anticipated by the expert and is not intended. Showing this unintended interaction between syntactically correct units can only be achieved in the eighth stage. This verification process is related to the problem of program verification, a well known and extremely hard research problem.

If undesired semantic and pragmatic interactions between pieces of knowledge were discovered in the eighth stage, then a ninth stage is necessary to respecify the tentatively added piece of knowledge. The flowchart in figure 5 summarizes the stages in the direct knowledge acquisition process.

1.2.2 Acquisition of Goal-Based Activity Knowledge

Goal-based activity knowledge is more complex than rule and skill-based activity knowledge. The different structures and levels of complexity of these

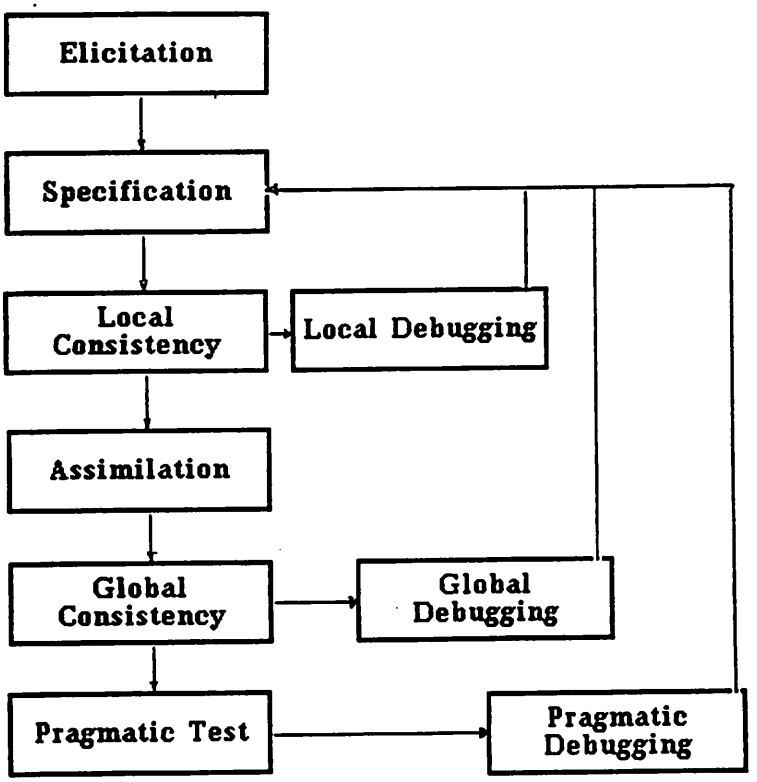


Figure 5: Knowledge Acquisition Processes.

types of activity knowledge have consequences for the application of the process model of direct knowledge acquisition.

Rules usually express a causal or temporal relationship. Goal-based activity knowledge (plans) usually consists of more elaborate information. Therefore rules are usually smaller in size then plans. This fact facilitates the elicitation and specification phase for the acquisition of rules.

Rule-based expert systems have to elicit the `IF` part of the rule and the `THEN` part of the rule. Once the experts have recalled the knowledge that *corresponds* to each of these parts of the rule, they can start to specify it. Techniques used to accomplish this elicitation are usually based on association, comparison or cueing [Boo84,MMW85]. Since plans have a more complex structure than rules (goal, subgoals, precondition, constraints, temporal and causal orderings, etc.), the elicitation process must be less uniform. Additional requirements for plan acquisition are a clear conceptual distinction between components of the plan, specific techniques to elicit particular components and an overall framework to integrate the components. Techniques developed for rule-based expert systems, like repertory grids, heuristic classification, certainty ranking or choice annotation are of limited help in this case. Certain forms of cued recall and structured frames seem to be most appropriate for the various demands in the acquisition of goal-based activity knowledge [MC89].

The largest problem in both areas is the specification of the elicited knowledge. The techniques applied most commonly are editors for the syntax of knowledge representation formalisms. This technique will not work as well in the acquisition of plan knowledge from domain experts because of the complexity of plans and the experts' lack of knowledge engineering skills. In planners, each

slot needs to be addressed in a semantically correct way that is also immediately understandable to the domain expert.

Specified knowledge can either be displayed in the same form that it was entered or it can be set into the context of other knowledge in the system. For rule-based expert systems, this usually means displaying a rule network. The different reasoning mechanism in planners leads to a different display strategy.

The above discussion shows that the acquisition of plan knowledge is considerably different from the acquisition of rule knowledge. In addition, there is no formal model linking the humans' subjective view of a task to a knowledge representation formalism.

1.2.3 Interfaces for Knowledge Acquisition

Experts in most domains are not computer professionals. They are interested in performing a job and, hopefully, achieving respectable results. When domain experts use computers they have a low tolerance for technical computer terms and opaque functionalities. An expert in a field has spent many years acquiring domain knowledge and will be reluctant to restructure or relabel his knowledge for the sake of knowledge engineering. It is therefore mandatory to allow direct knowledge acquisition in the language and from the viewpoint of the domain expert.

The implementational details of the underlying knowledge base should not be visible to domain experts, and the computer must not intrude upon them. Experts must be made to feel they are working on familiar tasks, not working with alien technology. At this point the direct knowledge acquisition problem

turns into a human computer-interface problem. Interface design and human computer-interaction become an integral part of the knowledge acquisition issue.

1.3 Applying Knowledge-Based Systems

Knowledge acquisition has no purpose in itself. The acquired knowledge must be used to enable purposeful actions by machines and people. The reasoning processes of the expert system as well as its results must be conveyed in a form immediately useful and understandable to the end user of the expert system. The expert system has to use its knowledge to aid users in the pursuit of their objectives in such a way that there are clear advantages to using an expert system.

1.3.1 Classes of Knowledge Application

Expert systems can use their knowledge in a number of ways to aid end users. On the one hand they can autonomously execute tasks and free the user completely of certain duties as in the case of autonomous robots. Expert systems can, on the other hand, work completely under the control of a human. Intelligent advice systems are an example of the latter.

The way expert systems are employed can be viewed according to the following three dimensions: activity execution, control, and dialog initiative. The first dimension concerns the actual execution of activities. The machine may only make suggestions as to which activities should be performed by a tool or by people. If the machine possesses effectors, like access to a database, or motors,

Table 1: Dimensions of Expert System Use.

Class of System	Execution	Control	Dialog
Intelligent Assistant	system	system	-
Exploration System	human	human	mixed
Intelligent Associate	human	mixed	mixed
Collaborative System	mixed	mixed	mixed

or network links, the machine itself might actually execute certain activities. A distribution of activity execution between the machine and the human is also possible.

The control dimension concerns the problem solving process. The expert system might start to solve a problem and return a complete or partial solution. In contrast to that mode, the machine might aid a human in a solving problem, where the human sets the course of action and the machine takes only single steps. A mixed mode, with the machine solving parts of a problem and the human filling in other parts is also possible.

The third dimension concerns the dialog between machine and human. This dialog might either be completely initiated by the machine with the human only responding to prompts of the system. It might be completely initiated by the human or it might be interactive.

Table 1 shows four different classes of expert systems rated according to the above dimensions. *Intelligent Assistants* autonomously perform tasks for human operators. Autopilots are an example of this class of expert systems.

Supporting human decision making in the search for an optimal course of action is shown as *Exploration Systems.* These systems allow their users to develop scenarios and try different strategies. By exploring the realm of possibilities for action, actual behavior of a person or an organization can be optimized.

Intelligent Associates are advice systems that transcend exploration systems by employing a mixed control strategy. The system proposes partial solutions and the human uses these solutions as advice for a particular problem at hand. The human informs the system about the evolution of the situation and the accomplishment of goals. Based on the new information further advice can be generated.

The most sophisticated expert systems are in the class of *Collaborative Systems.* Human and computer cooperatively develop plans to achieve goals and distribute the execution of activities among themselves. Initiative is taken by the human or the system whenever necessary.

Current interactive expert systems are mostly advice systems. In this dissertation we also deal with exploration systems. Research is under way to extend these systems towards collaborative task execution [CL88,LC89].

1.3.2 Knowledge in Context

When system and user interact to cooperatively solve a problem, knowledge must be communicated from the system to the user. The expert system has to make its advice, suggestions or actions understandable and clear.

The display of knowledge and its requirements changes with the class of knowledge at hand. Declarative knowledge, describing objects or facts has suc-

cessfully been dealt with in traditional databases. The form-based representation of information about objects and facts was adequate in fields like office information systems and data bases [Zlo82,Zlo75].

Activity knowledge is inherently different from declarative knowledge. Declarative knowledge is static and can be displayed in a static way. Activity knowledge, on the other hand, is dynamic and describes the changes in a situation. Communicating activity knowledge therefore requires techniques different from those for the display of declarative knowledge.

The classes of activity knowledge identified above, skill-based, rule-based and goal-based, add to the complexity of the problem for activity knowledge display. A rule, consisting of a precondition and an effect part, cannot be presented in the same way as a goal-based activity description, containing goals, preconditions, plan rationales, etc. Goal-based knowledge is more complex than rule-based knowledge and therefore requires different techniques for its communication.

What makes the problem of displaying activity knowledge even harder is its dynamic use. Sequences of activities may depend on each other and create a complex web of situations, interdependencies, constraints and changes. Instead of looking at the description of a goal-based activity in isolation, as an abstract entity, we have to present it as an integrated part of the purposeful behavior of an intelligent human or machine agent.

When the planner uses activity descriptions to satisfy a goal, knowledge is put into context. This context consists of higher and lower level activities, activities that happened previously, activities that can happen concurrently and

those that will happen later. The context also contains certain facts that could exclude other activities from being applicable at all.

To satisfy the goal of having a cup of hot coffee for example, one could go to the cafeteria and buy some. This activity requires money. If, in a certain scenario, one has no money, then buying coffee would not be possible. One could try the alternative activity of brewing coffee, which would also satisfy the goal. It does not require money. Instead it depends on the presence of ground coffee and hot water, and results in a cup of coffee, and some loss of ground coffee and hot water.

Activity descriptions that mention agents or objects in the abstract are also put into context when these abstract entities are bound to particular ones. Users must, for instance, understand that the company building a house is, in a specific case, bound to the *New England Home Builder Inc.* Using this particular company could result in a context that excludes other subcontractors or allows the choice of a large number of house types. These examples illustrate how context constrains the applicable activities and adds requirements to the display of activity knowledge.

In an advice, exploration or collaborative expert system, it is necessary to present the reasoning process of the plan and its results. The application of activity knowledge in context results in a sequence of activities that can be applied in the domain and will hopefully solve the problem. This sequence must be displayed and explained to the user. It could possibly be developed in cooperation with the user, but it must be clear to the user at all times what the expert system does and how the solution of the problem progresses.

1.3.3 Knowledge Display Interfaces

The requirements for the display of knowledge mostly concern display algorithms and techniques for the human computer interface. Users of knowledge-based expert systems are not interested in the system *per se* or its reasoning powers. Users are interested in getting their work done, in solving the current problems in the domain. Expert systems can be powerful tools for users to achieve these goals when they communicate their knowledge in a way that is immediately understandable to the users.

A variety of techniques have been developed for communicating information. Office information systems, management information systems, databases and graphical simulations are good examples for the use of such techniques. Techniques for the communication of knowledge and its dynamic changes are less well researched, yet such techniques are crucial to harness the power of expert systems for end users.

Since users are interested in the domain and not in the system, they should be provided with an interface that makes the system transparent. Such a system would let its user have the feel of directly dealing with the domain. Labels and terms would be the same as in the domain, processes and situations would evolve and be depicted according to the domain. Knowledge would be communicated directly and be immediately applicable to the user. Domain novices could acquire expertise, while experts could work quick and efficiently.

1.4 Dissertation Goals and Research Issues

The goal of this dissertation is to develop a highly usable system for the direct acquisition and display of goal-based activity knowledge, using an integrated cognitive engineering approach. We contribute to the technology and understanding of direct knowledge acquisition as well as to the communication of dynamic knowledge. We also develop a new method for building systems by integrating unrelated design, implementation, and evaluation approaches into a coherent methodology.

The major issues in this dissertation concern knowledge-based systems as well as users. Figure 6 shows the distinction between these two major concerns. Under each issue we find a number of more specific ones: knowledge acquisition, knowledge representation and applications on the system's side, and understanding the view individuals have of their own activity knowledge and the interaction of expert system and human on the user's side.

Each of these specific issues is rooted in a number of fields in computer science and cognitive science. Knowledge acquisition itself has recently been recognized as a distinct subfield of artificial intelligence. Planning and knowledge-based expert systems are closely related to knowledge acquisition. Knowledge representation also relates to planning and expert systems and contributes valuable constraints to our research goal.

Applications of knowledge-based expert systems with acquisition and display components can occur in traditional fields of computer science, like office automation and database. Techniques used in office automation, programming

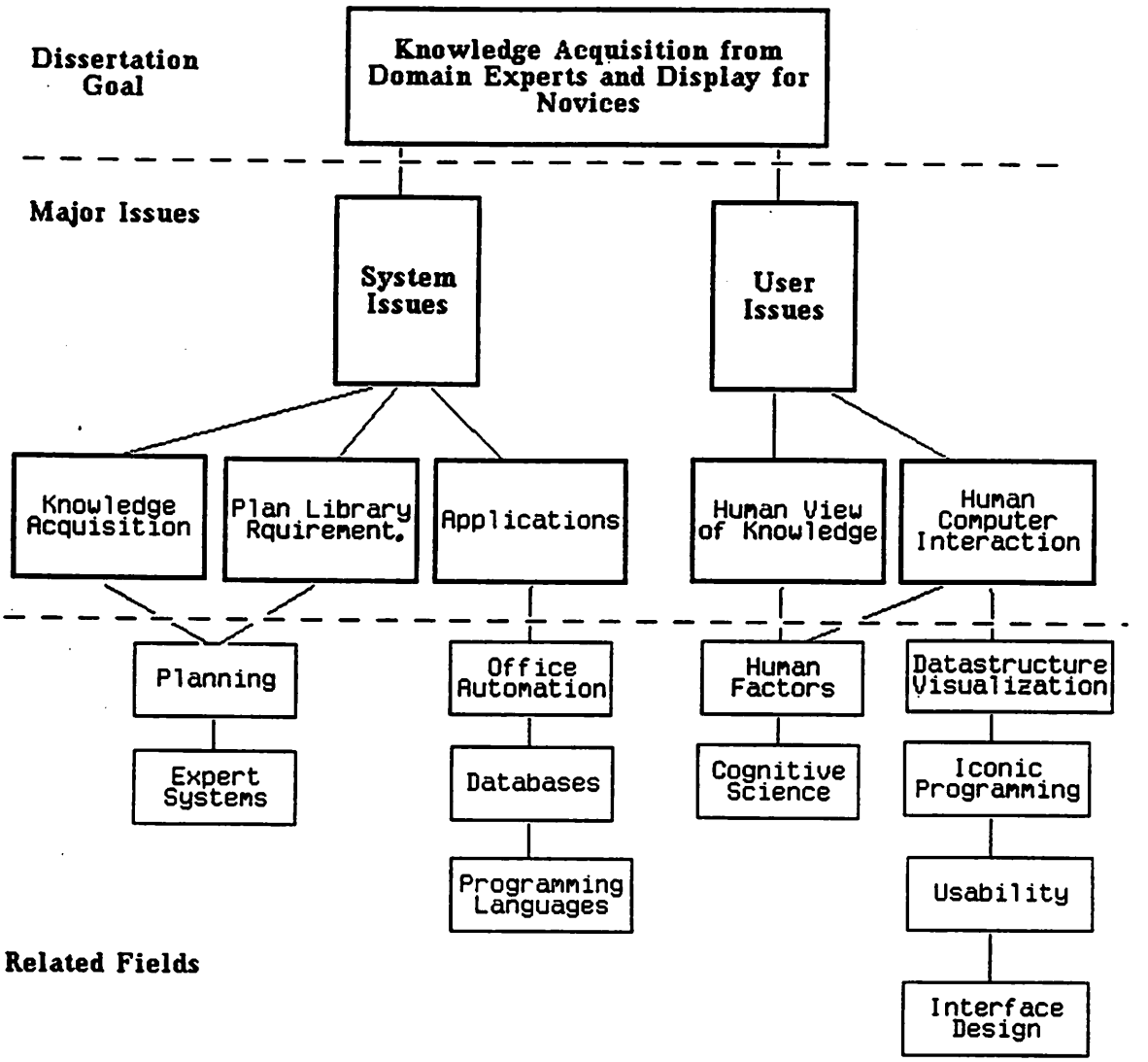


Figure 6: Research Goals, Issues and Related Fields.

languages and databases are potential candidates for the development of a knowledge acquisition and advice system.

Taking into account the view experts have of their own activities and relating it to the requirements of the knowledge representation formalisms is related to research in human factors and cognitive science. Human factors and cognitive science also contribute to our understanding of human computer interaction and the design of interfaces.

In order to build usable interfaces that are quickly accepted by users, we can draw on previous research in human computer interaction, particularly interface design and usability engineering. The visualization of data structures and advances in iconic programming are related to questions of knowledge specification and display methods for novices and experts. We will review the contributions of these fields to our research goal in detail in chapter 2.

1.5 Overview of Dissertation

In the course of our research we develop an extended version of the cognitive engineering approach [HW83,ND86,Ras86b]. We call this new approach *integrated cognitive systems engineering* and use it to build a system for the direct acquisition of goal-based knowledge and the display of advice. Our approach applies engineering principles to the design of systems which consist of cognitive subsystems such as knowledge-based systems or humans.

1.5.1 Integrated Cognitive Systems Engineering Approach

The integrated cognitive systems engineering approach starts by analyzing the cognitive and functional properties of each subsystem. The formal properties of plans yield a set of constraints for the construction of the knowledge acquisition system. Analyzing the experts' knowledge representation and the knowledge acquisition process provides additional constraints.

We introduce a theory of human task recall that integrates all of the above constraints and relates human knowledge representation and recall processes to properties of knowledge bases. Experiments involving more than 150 subjects are used to support the theory. On the basis of the theory, implications are drawn for knowledge acquisition and advice systems. Additional implications are inferred from our experiments and guidelines for interface design. The implications are translated to design specifications, which are implemented in the DACRON (Direct Acquisition of ConstRained Object Notation) system. Rapid prototyping, user participatory design [Gou89] and mathematical optimization of cognitive parameters [Fis88] are integral parts of integrated cognitive systems design during this implementation stage. The result of this process is a robust prototype that can be used for evaluation.

Usability testing concludes the integrated cognitive systems design approach [WBH87]. We conduct three studies with more than thirty subjects to evaluate usability goals concerning DACRON. The results of our usability studies show the effectiveness of DACRON and support our theory and the inferences in moving from theory to system.

1.5.2 *Mapping the Cognitive and Functional Space*

Following the integrated cognitive systems engineering approach we identify the the following common properties of goal-based expert system libraries: goals, preconditions, decomposition, effects, plan rationale and constraints. We also identify important properties of the knowledge acquisition process. Elicitation and specification of knowledge require the most interaction between human expert and system. We therefore concentrate on these processes in the design of our acquisition system.

The way domain experts view their activities is another source of constraints. To understand if domain experts can relate to plans, we need a theory that explains the way experts view and recall their view of the activities they perform. Except for experimental research on the recall of self-performed tasks [BNC86], we find no cognitive theories that make statements on a level required to relate human views of activities to the requirements of the planners and the knowledge acquisition process.

Due to this lack of directly applicable theories, we conduct interviews and questionnaire based research, which indicates the appropriateness of the concept of *Handlung* (approximate English translation: enactment). An enactment is the unit of behavior in action psychology, a field newly revived by American and German psychologists [KB85,FS85,VW85]. On the basis of action psychology and cognitive psychology, we construct an enactment theory for the recall of activities, which postulates that experts can recall the intention of an enactment, the situation in which it is appropriate, the steps that constitute it and the situation that arises after the steps are executed. The recall of an enactment

encompasses only one level of steps. To describe in detail how a step is executed experts change their frames of reference. What was previously a step becomes the focus of attention, the currently viewed enactment. Now the experts recall the steps of this new enactment. Objects used in the enactments can also be recalled, as can alternative ways to achieve an intention in a certain situation.

Paper and pencil experiments with 153 subjects, graduate and undergraduate students, support this theory. We detect that special care has to be taken with the recall of the situation after execution of the steps in an enactment. Subjects have difficulties recognizing the difference between the execution of a step in an enactment and the effects this step creates. Though the subjects are aware of the effects, they tend to perceive them as being contained in the step.

1.5.3 Designing the Acquisition and Display System

Employing traditional design techniques, such as weak deduction, we use enactment theory in a proactive way to derive implications system design. Proactive use of cognitive theories, using theory to design rather than to explain, has not been attempted on this scale before.

We observe a lack of strong formal techniques to generate initial designs during this phase of our research. The lack of qualitative methods to turn cognitive theories and guidelines into specific designs leaves the designer only with weak deduction and intuition.

Major implications for the knowledge acquisition and display system are the use of a dedicated editor for activities, reflecting the properties of the enactment and the labeling of steps, objects and other entities on the level of granularity

perceived by the expert. Visualization for *well formed formulae (wff's)* and iconic specification techniques are other important implications. These techniques are crucial, since planners capture their knowledge in wff's, which are expressions in predicate calculus. In this system, we restrict ourselves to wff's that consist of a predicate connecting two terms, for example, `(owns (road.to house) (lawyer.of peter))` expresses that the lawyer of Peter owns the road to the house. Figure 7 shows and editor for an activity with iconic wff's in it and an editor for an object.

1.5.4 Specifying an Architecture

Based on these implications we propose the architecture for a knowledge acquisition and display system, DACRON (Direct Acquisition of ConstRained Object Notation). The architecture is independent of the planner used as the back end.

In this dissertation, we use POLYMER as underlying knowledge-based system. POLYMER is a planner developed at the Collaborative Systems Laboratory of the University of Massachusetts [CL87]. POLYMER consists of a planning engine that uses domain specific knowledge bases to achieve goals. During the planning process, POLYMER constructs plan networks. These plan networks represent partial solutions to achieve the goal. Figure 8 shows a block diagram of DACRON, POLYMER and their connection.

DACRON creates and maintains a graphical representation of the planner's knowledge base. The knowledge base contains activities and objects and provides a collection of prototypical contexts for the execution of activities, called

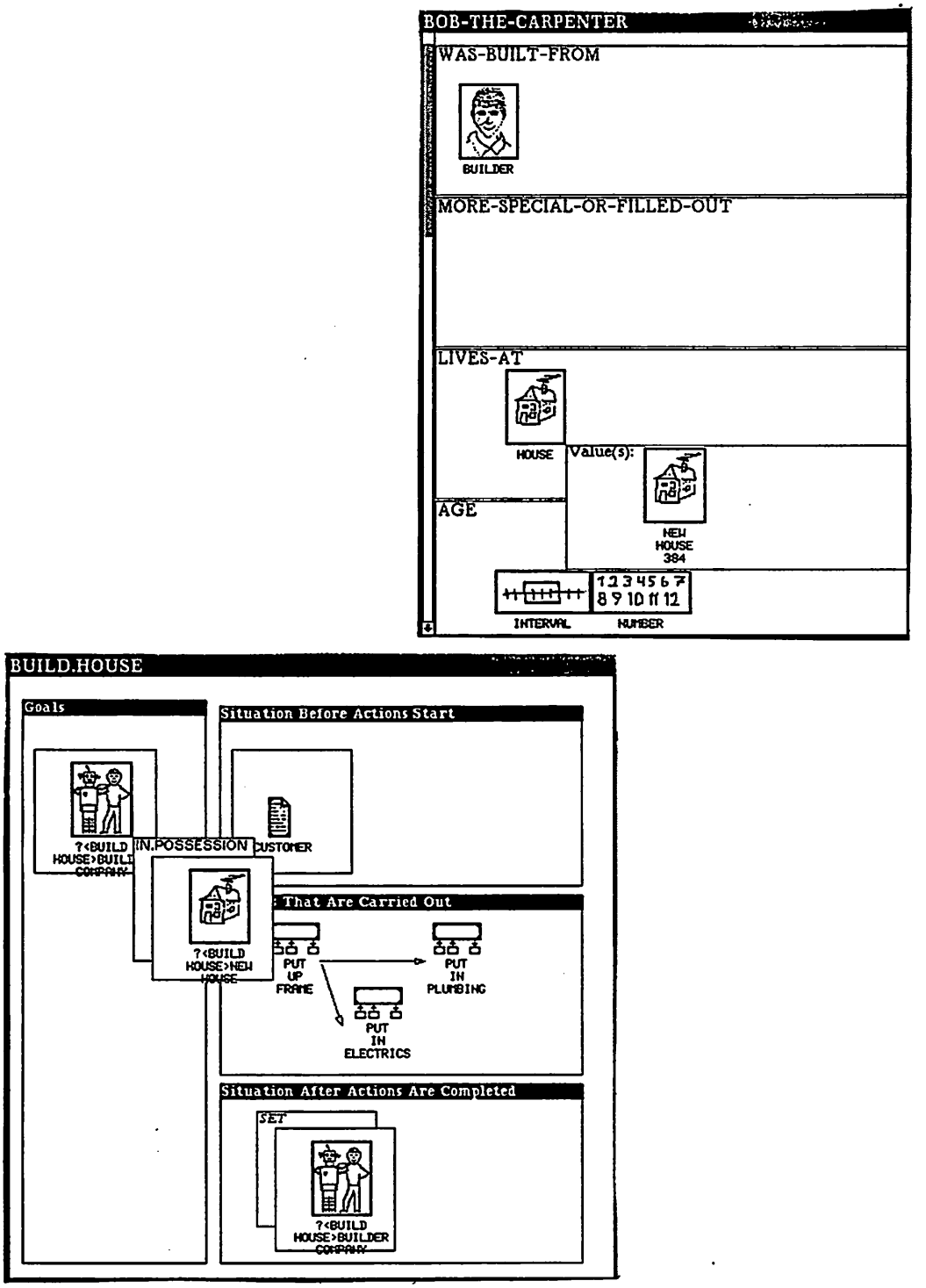


Figure 7: Activity Editor and Object Editor.

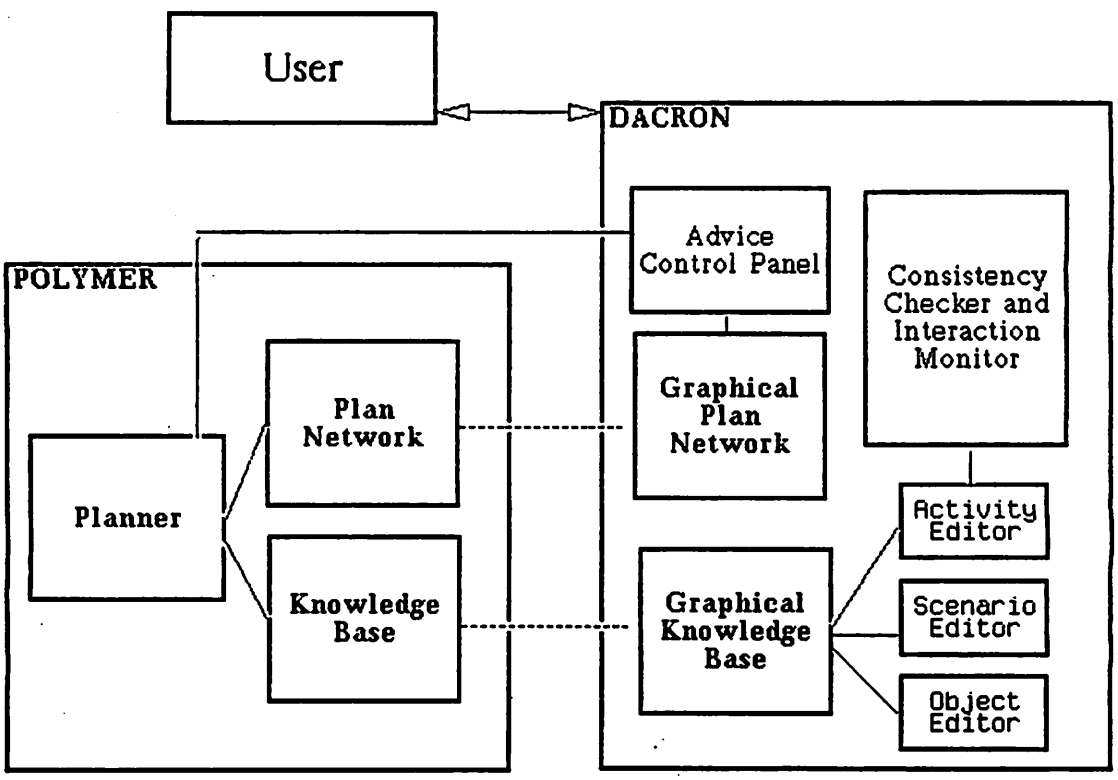


Figure 8: DACRON/POLYMER Block Diagram.

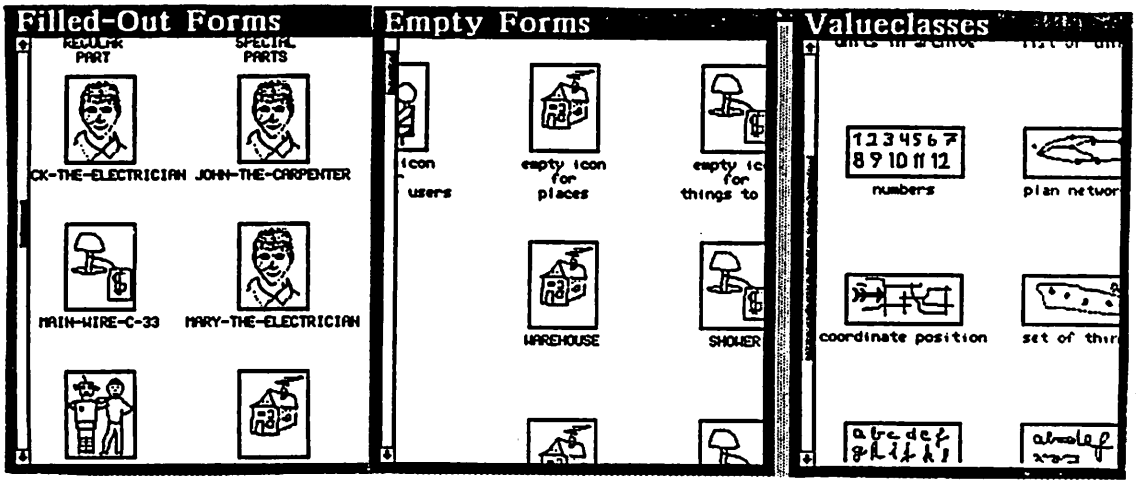


Figure 9: Windows representing Instances, Classes and Valueclasses.

scenarios. They are represented by icons in the graphical knowledge base. Special icon types exist for activities, scenarios and objects.

DACRON distinguishes between three varieties of icons in the knowledge base. Icons for classes are presented in one window, icons for instances of these classes in another. A third permanent window holds icons representing valueclasses, used for strong typing. Figure 9 shows the three windows.

The contents of an activity, object or scenario in the knowledge base can be viewed in a dedicated editor. There are editor types for activities, objects and scenarios. Editors can be used to display existing units or to specify new ones. Specification is achieved by copying icons from the graphical knowledge base to the appropriate place in the editor.

To display advice a control panel, invoking the planner, and a window graphically displaying plan networks generated by the planner can be manipulated by

the user. The planning process is visualized as a succession of plan networks. A plan network represents the flow of activities required to achieve the goal. As activities are replaced by more specific ones, the plan network grows and becomes more detailed.

Facilities exist to bind variables, to set parameters for the display process, and to navigate the knowledge base. A clipboard can be used to cache icons used frequently. Icons can be found by employing certain functions of the knowledge base navigation facility. The same facility allows the graphical reordering of the graphical knowledge base. A help window contains guidance during the use of the system. This window suggests the prototypical next user action, but does not restrict the actions. As suggested in figure 4, users work with DACRON and do not have to interact with the underlying planner directly.

1.5.5 Functionality

The simplest functionality provided by the system is browsing the knowledge base. Users can scroll up and down in the windows (see figure 9) or they can invoke a zoom facility to see the complete graphical knowledge base. Users can change the way icons are organized. There are algorithms to order icons by bitmap, alphabet or class hierarchy. Users can search for units or classes of units. This can be done by the name of the unit, or properties of the units such as a specific goal, a certain value, etc.

Icons can also be opened to display the content of the unit in a dedicated editor (see figure 7). Activity editors present the subject of a wff as a represen-

tative for the whole clause. The icon can be opened and the iconic wff appears. Constraints can be displayed on demand in the same way.

Objects are displayed as a collection of rows. The icon for the valueclass of that row is present. Opening it results in the display of the value. The object editor can be seen on the right in figure 7.

To specify a new instance of a class, users click on the icon for the class and choose the option `Fill out this Form` from the pop-up menu. A new object editor appears and the users are prompted in the editor for the name of the new unit. Users type the name of the new unit and fill the rows of the editor with values. The users can then save the editor and a new icon appears in the Filled Out Forms window

To create a new class of objects users seek the most appropriate object in the Empty Forms window. From its menu they choose the `Create a more specialized Form from this one` option. A new editor appears and the users are prompted for the name of the new class. The new editor has all the rows of the object it was created from. Now DACRON prompts the users if they want to add new rows to the editor. If users say *yes*, a row is added and users can label it.

Specifying an activity can either begin with updating an old activity, choosing the option `Create a more specialized Form from this one` from its menu, or with filling out an empty form for an activity. Figure 10 shows a new editor for the activity of building a house. To specify its wff's, icons are copied to the empty compartments.

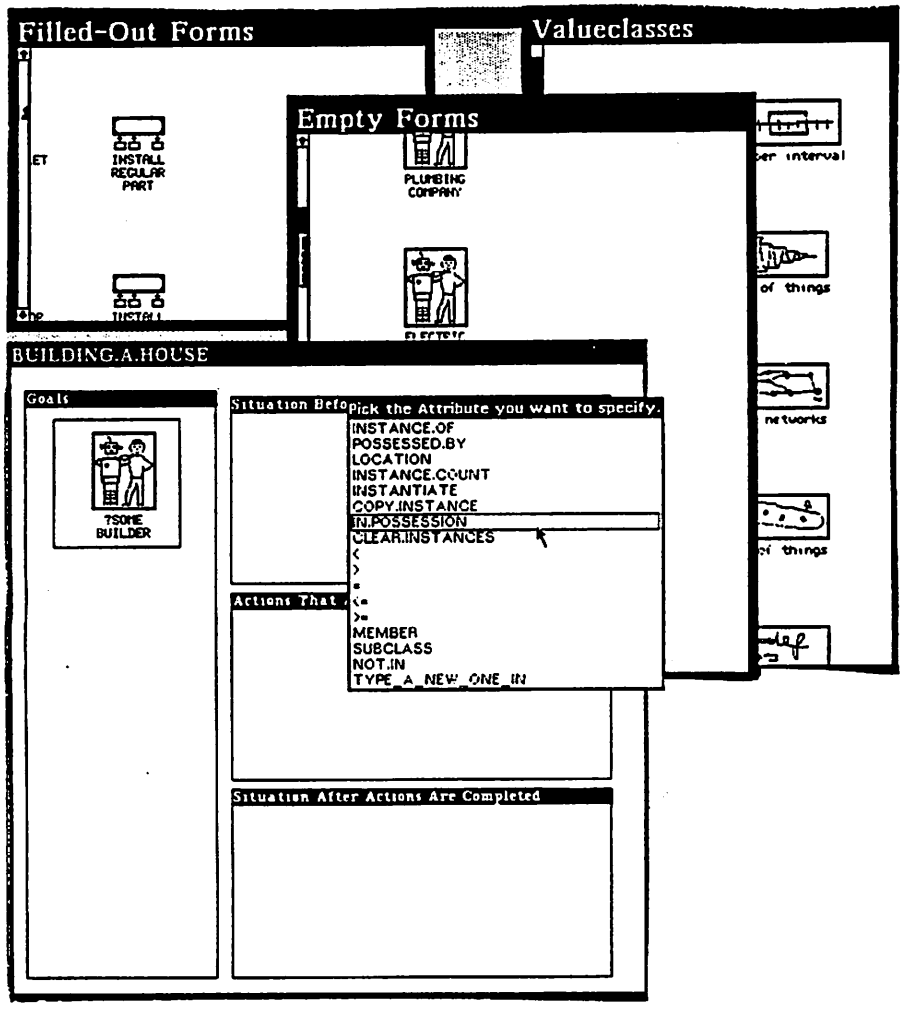


Figure 10: Specifying an Activity.

The goal in our example is that a home builder company be in possession of a new house. Users copy the icon for a specific home builder company from the Filled Out Forms window if they want to talk about a special company or from the Empty Forms window if they want to allow any home builder company. In our example the users copied from the Empty Forms window and named the resulting variable `?some.builder`. DACRON added the question mark to identify the icon as a variable. During actual planning in the advice stage, this variable will be bound.

To specify that a number of steps must be carried out to achieve this goal, users can copy icons representing these steps from the Filled Out Forms window. If no icon for the step can be found, users copy the icon for the empty activity from the Empty Forms window to the editor and give it the desired label. DACRON opens a subgoal holder and aids users in the specification of a wff to express the subgoal. DACRON also creates a new activity with the wff of this subgoal as its goal-wff. This guarantees that at least one activity exists that can satisfy the subgoal.

Advice is displayed when users choose the advice option from an icon for an activity. Users can then choose a scenario icon as context for the activity. Bindings for variables are set by the users by selecting appropriate values from a window of iconic choices, generated at run time.

Figure 11 shows the plan network being displayed in a window. Hitting the forward button on the VCR-like control panel makes the planner expand one of the goal nodes in the plan network. This nodes blinks (highlighted in the upper plan network of figure 11) and the activity used to expand the goal is displayed

in an editor, shown in the lower left hand corner of figure 11. Together with the editor, the refined plan network appears.

If additional variables must be bound or a choice must be made between alternative activities to achieve a goal, DACRON employs the same iconic choice method described above.

The refinement process is repeated until every node in the plan network is a primitive action, either executable by a person or a machine. If this is not possible the planner will halt, having generated a partial plan network.

1.5.6 Usability Studies

To verify the correctness of the design implications and the usability of the systems, we conduct an experimental usability evaluation. Usability engineering concentrates on the effective interaction between user and system. It goes beyond traditional evaluation which only measures system parameters. In the usability engineering technique, operationalizeable performance goals for interactions between system and user are defined. Experiments are conducted to see if the usability goals are met.

To demonstrate the usability and relevance of the acquisition and display system we specify a set of usability goals. These goals involve the elicitation of relevant knowledge from domain experts, the ease of specification, the relevance of the advice given, and the manipulability of the interface. Three experiments are conducted. Seven domain experts from the areas of energy auditing, architecture, and technical writing and and 25 graduate students from various departments participate.

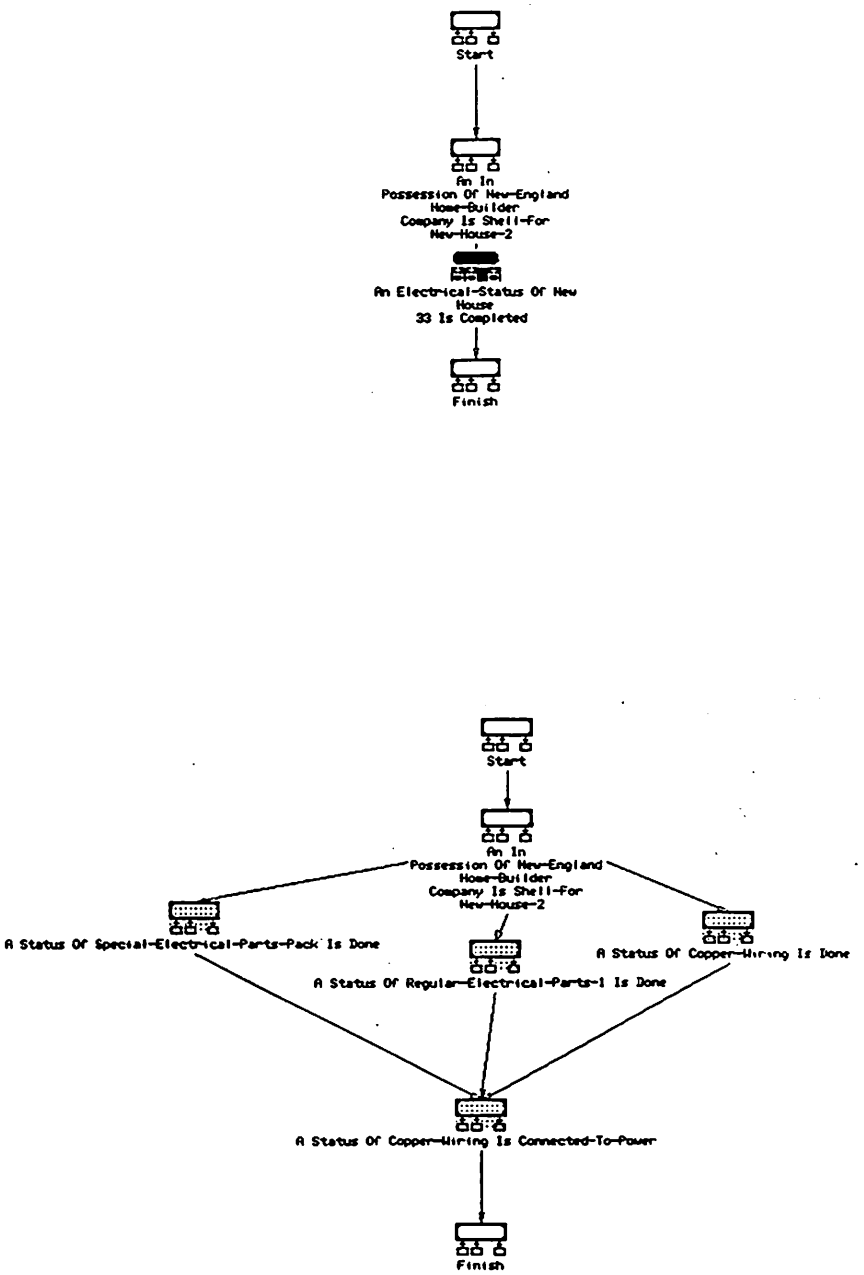


Figure 11: Graphical Plan Networks.

In the first experiment, 18 graduate students specify textual job descriptions using DACRON. Subjects are videotaped and the resulting knowledge bases are evaluated. All usability goals are met. Objects and activities are created with ease. Using knowledge specified by other users or updating such knowledge is accomplished quickly and correctly.

In the second experiment, 7 domain experts specify knowledge necessary to solve a problem in their domains. The same procedure as in the first experiment is applied. Knowledge is specified correctly. Some problems concerning the structuring of knowledge occur, but the produced knowledge bases can be used to offer advice to novices in the field.

The third experiment concerns the relevance and quality of advice given by the system. Graduate students and experts are presented with a questionnaire concerning a task that can only be solved using advice from the system. All subjects can use the system easily. Ninety-five percent of the questions are solved correctly.

CHAPTER 2

UNDERLYING REQUIREMENTS, THEORIES AND TECHNIQUES

In the previous chapter we defined research goals and issues. We now look more closely at their underlying systems and theories. This will provide us with a set of formal requirements for DACRON, our knowledge acquisition and display system.

We also review related fields that may provide us with techniques useful for knowledge acquisition and display. Guidelines and theories for the design of systems and interfaces are found by reviewing the field of human computer interaction.

We start with the knowledge acquisition problem. First we identify the requirements planners pose for the form and process of knowledge acquisition. From there we turn to previous knowledge acquisition systems to see what kind of techniques they employ and if we can use them for the acquisition of goal-based activity knowledge. Additional sources are knowledge elicitation techniques used in psychology and graphical code specification techniques.

Next, we review techniques and systems that apply and display knowledge. Visualization of data structures and graphical simulation seem to have potential for the display of knowledge, statically and dynamically. Office systems and plan

based advice systems are existing implementations that use activity knowledge to give advice to novices in a field. Office systems also emphasize interface design and ease of interaction.

Finally, to relate the human view of activities to the requirements of the planner, we review cognitive theories for the representation of activity knowledge. Continuing our focus on the user, we look for interface design guidelines and philosophies that can improve interfaces for knowledge acquisition and display systems. The evaluation of such system is of major importance, which is why we review evaluation techniques. Searching for a unified approach to integrate our research, we review cognitive engineering.

2.1 Knowledge Acquisition

2.1.1 Knowledge Acquisition Requirements for Planners

Acquiring plans for plan based expert systems requires that we understand the structure and the elements of these plans [CRM80]. We do not need to have an in-depth understanding of the conceptual differences between types of planners, but we need to recognize which elements of plans have become common and necessary for planning. This section will not review planning and control issues for landmark systems, but instead will identify common elements in the plan description languages of these systems.

Planners try to achieve a goal. Every plan formalism, from the very early ones [FN71,Sus73] to the most current planners [Ver83,Wil84] specifies a goal or multiple goals for the plan [FN71,Sac75,Wil84,Ver83,Tat76]. Goals express

the desired state of the world upon completion of that plan in a well formed formula.

The following example of a goal and the objects and variables used in it comes from the domain of constructing houses. A goal state in such a domain might be: *A construction company is in possession of a house.* This goal can be expressed as a well formed formula: `(in.possession ?construction.company ?house)`, where `construction.company` and `house` are variables, indicated by questions marks preceding the name.

All properties of plans are represented as well formed formulae. Well formed formulae, or wff's, are expressions in predicate calculus. Wff's make statements about terms or objects, represented as constants, variables or symbols. A wff consists of a predicate defining the relation between terms. In the above example, for instance, `in.possession` is the predicate relating the variable `?construction.company` to the variable `?house`.

Most planners use binary predicates but n-ary ones are possible in predicate calculus. An example for a tertiary wff would be: `(gives peter mary tennis.racket)`, to express that Peter gives Mary a tennis racket. Wff's can also be constructed using the connectors `and`, `or` and `not`. An example for such a wff would be: `(and (in.possession ?construction.company ?beams) (in.possession ?construction.company ?sheetrock))` to express that the construction company is in possession of sheetrock and beams. Additional complexity can be achieved by adding the quantifiers `some` and `all`. An example for a term employing a quantifier would be `(mortal (all humans))`.

Plan languages, expressed in predicate calculus, allow for the specification of preconditions. Preconditions are expressions that have to be true in order to make the plan executable. These preconditions can be viewed as required instantiations of a subset of the knowledge base schema; in other words, parts of the knowledge base have to have certain values or the plan can not be executed. Traditionally, preconditions are expressed as predicates on objects, e.g. `(status (credibility ?construction.company) good)`, expresses that this plan can only be applied if the credibility of the construction company concerned is good.

The same is true for the expression of effects. Effects describe changes in the world that are brought about by the activity, but that are not the goal of the activity. Effects happen also but the activity is not performed for their sake. They can be viewed as ADD lists and DELETE lists or as secondary and tertiary goals. The change of the knowledge base through the application of plans must be specified in the plans. Systems like ABSTRIPS have two lists, an ADD list for additions to the knowledge base and a DELETE list for deletions [Sac74]. Other systems combine these changes into one EFFECTS clause. The EFFECTS clause can express both the ADD and the DELETE clause, as well as other effects, such as SET. The changes specified in the EFFECTS clauses translate into updates on the knowledge base.

Example from the construction domain might be: `(SET (status ?house complete))` to indicate that house under construction is now complete, or: `(DELETE (in.possession ?builder.company ?house))` meaning that the fact that the builder company is in possession of the house is no longer true when the plan of selling the house is executed.

The decomposition of the goal into subgoals is another common element of plans. Differences exist in how these subgoals are viewed and achieved, but the notion of decomposition into "steps" through goal matching is the very heart of planning [Sac75]. The plan language has to provide a consistent and perspicuous formalism in describing the goal of a plan and the subgoals it can be broken into. The subgoals have to be unambiguously identifiable. The subgoal in the higher level plan and the goal in the lower level plan have to be identical. In most plan languages this is done by naming or the matching of wff's.

The goal that a builder's company is in possession of a house, as mentioned above, can be decomposed into the following subgoals:

- the company is in possession of a frame;
- the electrical wiring is completed;
- the plumbing is completed.

These subgoals in turn, can be expressed as wff's.

Constraints limit the values variables can take. They also express dependencies among variables used in a plan. These constraints differ from static consistency constraints as found in data bases. Constraints in plans are dynamic. They govern temporal bindings and restrictions. Like preconditions they are usually expressed by wff's.

To ensure that the variable `?house`, used in the above examples, is only bound to a value that is of type `HOUSE`, a wff like: `(member ?house (class HOUSE))` should appear in the plan. Another example regards the interaction of bindings within one plan, stating that the weight of the beams for a frame should not exceed the weight of the roof: `(< (weight ?beams) (weight ?roof))`.

The temporal ordering of subgoals is also common to most modern planners. It is often referred to as the *plan rationale.* It basically specifies the order of execution among the subgoals. The plan rational determines which subgoals can be achieved concurrently and where strict sequential order is necessary.

The subgoals mentioned in the example above clearly can not be achieved in at the same time. The following ordering is mandatory: `(enables have.shell do.electrical.work) (enables have.shell do.plumbing)`. This plan rationale indicates that the shell must be completed before plumbing or electrical work can be done. It makes no statement about the order of the electrical work and the plumbing, therefore allowing them to occur at the same time.

To summarize, we can say that the following elements can be found in most modern plan formalisms:

- goals;
- decomposition into subgoals;
- preconditions;
- effects;
- plan rationale;
- constraints.

POLYMER, a plan based system developed at the Collaborative System Laboratory at the University of Massachusetts, is a modern planner and contains all the above elements. To tie our research into knowledge acquisition and knowledge display to a real system, we use POLYMER as a prototypical planner. All the concepts and results introduced in this dissertation can not only be applied to POLYMER but to any planner that meets the standards outlined in this section. A detailed description of POLYMER follows in section 2.2.4.

2.1.2 *Knowledge Acquisition Systems*

Knowledge acquisition in the area of expert systems has been mostly concerned with the acquisition of concepts, hierarchical structures between concepts and cause-effect relationships. Frequently, more attention has been paid to the resulting expert system and its inference engine than to the actual knowledge acquisition process. The assimilation and the debugging stages of the knowledge acquisition process, as shown in figure 5, have often been given more emphasis than the elicitation and specification of knowledge.

The existing knowledge acquisition systems differ in the domain to be acquired, the presentation of data, the amount of the systems inference, the techniques employed to elicit the expert's knowledge, etc. A number of classifications for these systems and their properties exist but none is widely accepted.

Table 2, which is an augmentation of the categorization work done by Boose [GB88], summarizes and relates the major knowledge acquisition systems. For reasons of completeness, we include DACRON, the knowledge acquisition and display system introduced in the remainder of this dissertation. The table consists of 19 attributes. Each of the 23 knowledge acquisition systems is rated on each attribute. The highest rating, 5, means the systems possesses the property on the right hand side of the table completely. The lowest rating, 1, means the systems completely possesses the property on the left hand side.

Table 2: Repertory Grid Summarizing Knowledge Acquisition Tools.

Current	1 3	2 5 5	1 4 1	1 1 1	1 1 1	1 1 1	1 2 1	2 1 1	Old
No Text Analysis	1 1	1 1 1	1 1 1	1 1 1	1 5 1	1 1 1	3 3 5	1 1 5	Text Analysis
No Grids	1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	5 5 5	5 5 5	Gridbased
No Protocol Anal.	1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	5 5 1	1 1 5	Protocol Analysis
Analysis	4 5	1 1 1	1 1 1	1 1 5	3 1 2	1 1 1	1 1 1	1 4 5	Synthesis and Anly.
No Decis. Anly.	1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 5	1 1 1	1 1 5	Decision Analy. Meth.
Not Stat. Domain	3 1	5 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 5	Statistics Domain
No Learning	1 1	2 1 2	2 1 1	1 2 1	1 1 1	1 1 1	1 1 1	1 3 5	Learning Capabilities
Rigid Uncert. Meth.	1 1	1 1 1	1 1 1	1 1 1	1 5 5	1 1 1	1 1 1	1 5 5	Flex. Unc. Meths.
No Induction Tools	4 4	2 4 2	4 1 1	1 2 1	1 1 1	1 2 1	1 1 4	4 5 5	Induction Tools
Few Knowl. Views	4 1	2 1 2	3 2 2	2 1 2	2 1 1	3 3 2	1 4 4	4 5 5	Mult. Knowl. Views
Few Tools	3 2	3 2 3	3 2 2	3 1 2	2 1 2	1 2 1	2 3 4	3 5 5	Many Tools
No Interviewing	3 1	1 3 5	5 3 3	3 1 2	3 2 1	1 1 1	2 4 5	5 5 5	Interviewing Tool
Copy an Edit	2 5	3 4 4	5 4 4	4 2 2	4 3 3	1 2 3	3 4 5	5 5 5	Higher Level Tech.
Manual	2 5	3 4 5	5 4 4	4 2 2	4 3 2	2 2 1	3 1 4	5 5 5	Automated
Domain Spec.	5 5	1 3 5	3 2 2	2 1 1	2 2 4	5 5 5	5 5 5	5 5 5	General
Deep Modeling	1 3	2 4 3	1 2 2	2 2 1	3 4 5	5 5 4	5 4 4	4 3 1	Shallow Modeling
No Diagnostics	1 1	1 5 5	5 5 5	5 5 5	1 1 4	3 1 1	1 2 4	4 4 5	Diagnostics
Medical Diagn.	3 3	1 1 1	5 5 5	5 5 1	1 1 1	3 3 1	3 3 3	1 1 1	Mechanism Diagn.
Tool Reference	D K	S R T	A B C	E F O	G H I	N Z P	Q U V	W X Y	See Legend

Legend

D - DACRON	K - KNAC	S - STUDENT	R - Roget	T - Teiresias
A - MDIS	B - MOLE	C - MORE	E - TKAW	F - FIS
O - OPAL	G - SALT	H - KNACK	I - MUM	N - NEXPERT
Z - KREME	P - INFORM	Q - KRITON	U - PURDUE	V - KITTEN
W - ETS	X - AQUINAS	Y - Ideal Solution		

Knowledge acquisition systems rely mostly on linguistic transmission of knowledge. Interview and protocol analysis must be applied through a knowledge engineer. Domain experts cannot interact directly with the system. The same is true for task analysis.

Grids as introduced by Boose [Boo84] and questionnaires/forms as used by Musen [MFCS86] allow domain experts to shortcut the cumbersome process of interviewing and give them the possibility for transferring their knowledge directly to the system. While grids and triplet distinction allow the construction of elaborate concept hierarchies, and the limitation of state spaces for condition or effect specification, they cannot be used for the operational specification of activities. The questionnaire/form approach does not carry these limitations.

There are some knowledge acquisition systems that acquire "plans" [MFCS86] [MMW85]. In the context of these systems, "plan" means a skeletal procedure but not the goal-based description of activity knowledge. Still, these systems are the closest existing knowledge acquisition systems to our research goal.

ONCOCIN acquires treatment plans for cancer therapy from expert physicians [MFCS86]. Treatment plans can be operationally specified by using electronic forms and questionnaires that are closely adapted to the domain and the notational conventions used in cancer treatment. The plans acquired by ONCOCIN are only skeletal plans. They do not embody the concept of subgoal decomposition. One can think of skeletal plans as procedures with variables. In giving advice ONCOCIN just needs to select the appropriate plan and bind the variables correctly. Hierarchical planning, in the sense of section 2.1.1, does not happen.

SALT is a similar system. SALT is concerned with the acquisition of skeletal plans in the domain of elevator design [MMW85]. SALT's strong point is the integration of the problem solving method *propose-and-revise* with the acquisition of knowledge. Knowledge is entered in a menu-driven way. The major units of acquired knowledge are named procedures and constraints. Experts can enter knowledge in an unrelated way, while SALT tries to fit these pieces together. This process leads SALT to the detection of gaps and constraint violations. SALT then revises its initial design and elicits missing information from the expert.

AQUINAS is a successor to the initial grid based tools, like ETS or KITTEN [BB88]. AQUINAS is a collection of grid based tools, each tool dedicated to a particular class of knowledge. First, AQUINAS elicits distinctions. A grid is presented and the expert fills in the possible solutions to a problem in the columns and the properties of these solutions in the rows. The expert then has to fill the cells in the grid, ranking how each possible solution satisfies the property. AQUINAS helps the expert do this ranking task by offering triadic elicitation techniques and corner filling methods. Then the initial problem has to be decomposed. This is mainly done by giving the expert tools to create hierarchies of solutions, of traits and of other sources. Eventually the acquired knowledge is tested in a mixed initiative fashion.

KREME, like AQUINAS, is a collection of multiple tools [AB88]. KREME is not based on the grid paradigm but is rooted in the design of knowledge engineering environments. KREME offers editors for rules, frames, procedures and functions. These dedicated editors can be used on top of knowledge engineering environments such as LOOPS, KEE, etc. It also provides different views

of the same knowledge. New pieces of knowledge are classified and integrated, but there is no dedicated elicitation component or user model. A consistency checker verifies the resulting knowledge base. Knowledge is acquired by experts using the ZEMACS-like knowledge editors.

We find no existing system that acquires goal-based activity knowledge or uses a cognitive model of the experts' view of knowledge. We will therefore have to turn to the underlying scientific and engineering fields to find methods and theories for the acquisition and display of goal-based activity knowledge.

2.1.3 Knowledge Elicitation

We identified elicitation to be the first stage in the direct knowledge acquisition process. To review the available techniques for the elicitation of knowledge, we turn to the behavioral sciences, psychology in particular, since it is the science of human behavior.

Experimental and cognitive psychology are based on data derived from human behavior. The need for all kinds of human behavioral data is essential to these disciplines. An array of methods and techniques has been developed to obtain these data [Sch88].

There are techniques that require an observer or interviewer, like the following:

- observation of free behavior;
- observation of constrained behavior;
- free interview;
- structured interview;

- introspection;
- free association;

Then there are techniques that employ sampling by a machine:

- reaction time;
- frequency of errors;
- free recall;
- cued recall;
- recognition.

Other techniques are paper based, like questionnaires or tests and grids.

Items on these lists are not necessarily on the same methodological level. Some techniques are specializations of others. More techniques might be generated by combining some of the techniques.

Some techniques can immediately be disregarded for the purpose of eliciting the subject's activity knowledge in goal-based terms. All forms of observation, association and interview require an observer and a sophisticated task or protocol analysis.

Introspection is a method that was heavily used by early psychologists, then condemned by the behaviorists, and is currently undergoing a renaissance. The basic assumption of introspection is that one can observe one's own cognition at work. Apart from the apparent methodological controversy, introspection is not useful for our purposes because it is concerned with cognitive processes and structures. We are interested in content and reportability, not in the subjects' ability to look at their own modus operandi.

Reaction time and frequency of errors are the most rigorous methods for obtaining data in cognitive psychology. Unfortunately, these methods convey no semantic meaning in themselves, and need to be interpreted. Goals, preconditions, etc., can not be extracted directly from reaction time or form errors.

Questionnaires, tests and personality grids are very structured methods that have been used with success in the acquisition of declarative knowledge and rule based activity knowledge [BG86,BG87]. This success was possible because of the relation between psychological scaling, which is the underpinning for all clinical tests and personality grids, and the need for hierarchies of concepts in expert systems. It is not possible, however to operationally specify the complex contents of goal-based activities by these means.

Free association is a technique used mostly in psychoanalysis. Given a stimulus (a blob in the Rohrschach Test; a theme of a previous therapy session; a word fraught with emotions; etc.), subjects are asked to report associations elicited by the stimulus. These associations may become new stimuli and elicit new associations until a whole chain of associations grouped around a central problematic theme of the patient is generated. The data obtained by free association are usually interpreted in a certain psychoanalytic framework. This technique is not necessarily restricted to clinical use. In combination with recall, details might be elicited by cueing the subjects with stimuli central or peripheral to a certain task.

Recall and recognition are very general phenomena used to obtain data, mostly in memory experiments. A recall task usually consists of a list of verbal or geometrical items that subjects have to memorize and a recall phase in which subjects are asked to reproduce as many items from that list as possible. Recog-

nition tasks consist of the same stimulus material in the memorization phase, but in the test phase subjects are also given stimuli which were either present in the first list or not. In the case that they were in the first list, subjects are to respond with *yes*, otherwise with *no*.

Baeckman, Nillson and Chalom extended the recall paradigm from declarative knowledge to activity knowledge [BNC86]. Instead of recalling objects, subjects were to recall small activities they had previously performed. A recall list would look like this: *threw a ball, chewed gum, drew a house, lit a match, etc.* The results of these recall experiments showed that the recall of activity knowledge is qualitatively different from the recall of declarative knowledge. Subjects recall more of the activity knowledge with fewer errors than they recall of declarative knowledge.

This method seems appropriate for our purpose, because it is generally concerned with retrieval of activities. Expressing activity knowledge is not a form of problem solving. It is simply the reporting of knowledge stored in long term memory. This process can be viewed as a form of recall. It is not recognition, because there are no description of tasks we could give as comparison stimuli. The major difference between orthodox recall and the sort of recall we are interested in concerns the nature and the complexity of the material.

2.1.4 Knowledge Specification

The specification of knowledge resembles programming activities. Ideas for knowledge specification can hopefully be found in the fields of end user programming and visual programming.

Initial attempts to make programming easier consisted of systems that allowed programmers to manipulate orthodox graphical notations, like flow charts, transition diagrams or E-R diagrams directly.

PICT D is an example of a system that models a flowchart. It is a version of PASCAL that works like a jigsaw puzzle [GT84]. Commands and data structures are picked with a pointing device from a menu window. The keyboard is only used to name variables or procedures. A graphical editor allows the creation of new graphical symbols for procedures.

State transition diagrams are a more orthodox form of visual programming. By labeling states and transitions, programs can not only be documented but written [Jac85].

Database programming by manipulating screen versions of E-R diagrams in a straightforward way can be found in QBD [ACS89], Modeller [Dud89] or the work of Rogers and Cattell [RC88]. Rectangular boxes for entities can be copied and edited through mouse operations. Diamonds, representing entities can by manipulated in the same way.

The initial attempts to make programming easier by letting programmers manipulate flowcharts or E-R diagrams directly were soon replaced by more sophisticated visual techniques. Great interest in a more sophisticated visual programming paradigm was triggered by the appearance of SMALLTALK [Gol84]. SMALLTALK allowed the quick creation of graphical objects that could be treated in a sophisticated way in the object oriented approach. SMALLTALK sparked a lot of interest in the potential of visual and iconic programming [GF84,Smi86,Gut87].

A graphical extension to LISP is PAM [Lak80]. A set of primitive graphical objects (character and drawline) is introduced that can be manipulated with the usual LISP commands. From these primitives more complex graphic patterns may be created.

A system that allows programming by demonstration, not to be confused with programming by example, is ThinkPad [RGR85]. This system provides a dynamic programming language based on data abstraction. Data structures are designed by drawing appropriate graphical representations. Functions for those structures are defined by editing the representations.

Iconic systems provide a more drastic departure from the traditional programming paradigm. These systems allow direct manipulation of the graphical elements in the language. In this category are Pygmalion [Smi77], Thinkertoy [Gut87] and many others.

HI-VISUAL is a true interactive iconic programming language [HIY+87] [HYTI89]. Icons represent objects that exist in the real world, although most work has been done with office objects like papers, sales books and calculators. These icons have both data and functional properties. The function of an icon is not fixed but determined at the time of programming in relation to juxtaposed icons. Using the cursor users can move and overlap icons, thereby creating programs. The sequence of user actions is recorded in frames that can sequentially be executed, producing a sequence of frames as in a film.

A recent attempt to construct an object oriented visual programming language exists in Prograph [CGP89]. Prograph includes the main concepts of object oriented programming and dataflow. Classes with attributes are created

by copying class-icons from a graphical representation of the tree of classes. The icon for the class can be opened and augmented. Special icons to represent operations exist. Connecting icons for classes and icons representing operations by lines constitutes the programming process.

Apart from actual visual languages and iconic system, we find more and more environments and tools for the construction of such systems. These environments provide icon editors, graphical parsers, facilities for program development and system navigation and the integration of existing subsystems. Examples for such efforts are the work on environments for HI-VISUAL [HIY+87] or the work of Clarisse [Cla85]. The systems achievable with these environments are still very restricted.

We conclude that the iconic specification of knowledge could be achieved with current programming techniques. Our review also shows that iconic programming systems are welcomed by all classes of users.

2.2 Applying and Displaying Knowledge

Our second issue is the use of the acquired knowledge to generate help and advice for users. We start by looking at techniques that can be useful in visualizing knowledge. To deal with the dynamic planning processes, we review the methods used in the graphical simulation of processes.

Office automation systems have used activity knowledge for some time to aid office workers in their pursuits. Reviewing this field, we hope to find techniques for applying activity knowledge that can help us make planners easier to use.

Planners have been used only for a short time as advice systems. We review the results of these efforts to gain insights into the problems of using goal-based activity knowledge in providing advice.

2.2.1 Graphical Representation of Data Structures

Knowledge bases that store the representations of plans and objects are related to data bases. As data bases have a longer tradition than knowledge bases, more research in the visualization and usability of data bases has been conducted.

In the *Spatial Data Management System* [Don78] structures in the database are organized in three perceptual spaces: visual, auditory and haptic. Data are represented and manipulated in these spaces. A special cluster of input and output media was constructed especially for this purpose. The user flies like a pilot through the three dimensional, visual database. As he passes certain objects he hears their associated sounds and is able to maneuver towards them. Locations may be remembered haptically be sensing the position of the hand on the joystick. This approach draws heavily on qualities of interactive systems that may be a priori understood by the user.

In TINKER [Lie84], data structures in LISP, like lists and trees, are represented graphically. The screen is divided into windows which show the LISP code and the graphical representation of the active data structures. Running the code gives a graphical view of the change in the data structures. The data structures are represented as nodes of values linked by edges. The rationale behind this approach is the observation that programmers scribble diagrammatic

graphs when writing or explaining programs to visualize the connection and change of data structures.

A more general approach was taken in the APT (a presentation tool) system [Mac86]. APT is incorporated into an intelligent interface and constructs graphical representations of data, like charts, bar graphs, scatter plots, etc., automatically. This is with only a few primitives in the system. The information is divided into components, each of which satisfies an "expressiveness criterion" for a primitive graphical design. On the basis of a conjectural theory of human perception composition operators are applied to compose individual designs into a unified presentation.

INCENSE [Mye83] associates an "artist" with each data structure. This makes it possible to graphically inspect data. The artist knows how to draw the data, thereby producing a graphical representation of the relation of data. INCENSE is restricted to the PASCAL-like MESA language.

2.2.2 Animation and Graphical Simulation

The second part of our research problem addresses the use of the acquired knowledge. This involves the display of the acquired knowledge and the presentation of the planning processes. These problems are related to the fields of animation and graphical simulation.

In this context we distinguish between algorithm animation, used to visualize the execution of an algorithm, for example quick-sort, and the animation of graphical entities. This second kind of animation is similar to the process employed in creating cartoon movies. A computer based, graphical representa-

tion of one or more real world objects (a face, a person, a group of animals) is manipulated to create the illusion of motion (facial expressions and lip movements, running and walking, flight from predators). The object is represented as a data structure and an algorithm is used to compute the desired changes in appearance.

These two kinds of animation must also be distinguished from the graphical simulation of real world systems. In the graphical simulation case we have a program that models some behavior. To give the user of this simulation program an understanding of the change and interaction of the parameters in the simulation, graphical techniques are used. Because there might be many parameters that change quickly in time, sophisticated data representation techniques and methods to show interaction between the parameters are needed.

Balsa [BS84] and its successor at Brown University are the best examples of systems that give an animated view of the behavior of algorithms. Other systems worth mentioning are ThingLab [Bor79] and Duisberg's work [LD85]. All these systems try to emulate an effect that was first created in the famous color sound film *Sorting out Sorting* by Baecker [Bae81].

Systems that animate representations of human figures are mostly hand coded sequences of computer graphics. Interesting work has been done on matching the representation of a face and its lip movements to speech [Nis86]. Another line of research has tried to visualize the behavior of autonomous agents and their reactions to one another's movements [LW87]. Graphical objects are assigned behavior functions which make them appear to behave purposefully.

The graphical simulation of technical processes comes closest to the visualization of plans and the planning process we are interested in. SIMSCRIPT [CAC] and GRAPES [Mes86] are examples for these sort of systems. SIMSCRIPT is a commercial product for PC's that allows the creation of simulation programs and the iconic representation of the simulation results. Therefore SIMSCRIPT is limited to operations research-like problems. GRAPES provides a user friendly environment for the development of simulation models using block-oriented simulation languages. This block orientation severely restricts the usability of GRAPES.

2.2.3 Office Automation and Office Information Systems

The automation of offices is just about to merge with the programming efforts outlined above. Office automation is an obvious application for systems that give task support and therefore need descriptions of these tasks and methods to convey advice or assistance. This task support is mostly given in the context of office documents integrated in databases. Since we are interested primarily in tasks, we will review these database systems only as far as it is necessary for the specification and execution of office procedures.

Most office automation systems carry a procedural flavor. Examples are ODL [HHKW77,HHW74] or the integrated office system of Zisman [Zis77]. These systems were not graphical or visually oriented, but define actions and objects through alpha numeric keyboard input. The commands and structures in these higher level, application-centered office description languages could be viewed as procedures in integrated mail and filing systems. A deviation from this tradition, but not in the direction of direct manipulation or visualization was taken by OBE

[Zlo82]. The graphical structures used in OBE are two dimensional arrays with fill in slots, which provide cross referencing between various data structures.

OFFICETALK ZERO [NM83] is an integrated experimental office system. It integrates a large number of office functions including document preparation, illustration, electronic mail, filing, etc. OFFICETALK ZERO provides a unified interface of the mouse, menu and window type for these constituent tools.

The POISE system, developed at the University of Massachusetts, centered on the task support for office workers [CL84]. POISE was designed to transcend functionality provided by generic office tools such as calendars, forms packages or mail through the support of the problem solving abilities for the office worker, and assisting in complex sequences of actions which did not correspond to tool invocations.

POISE could be used to automate routine tasks and to provided assistance in more complex situations by performing a simple form of plan recognition. The type of assistance provided could range from maintaining a record of the tasks currently being executed to suggesting possible next steps and answering natural language queries about tasks.

The POISE system provided task support on the basis of hierarchies of task descriptions, where tasks were treated as procedures. The formalism to represent those procedure descriptions was a modified version of an Event Description Language [Bar83] . Figure 12 shows a purchase task expressed in that formalism. The IS clause specifies the syntax, which in turn is refined by the COND clause. Parameters are defined in the WITH clause. Conditions for a procedure to

```
PROC            Purchase_Items
DESC            (Procedure for purchasing items with non-state
                 funds.)
IS              (Receive_Purchase_Request
                 '(Process_purchase_order |
                   Process_purchase_requisition)
                  '(Complete_purchase))
WITH            ((purchaser =
                  receive_purchase_request.form.purchaser)
                  (items = receive_purchase_request.form.items)
                  (vendor_name =
                   receive_purchase_request.form.vendor_name))
COND            (for_values {purchaser items vendor_name}
                  (eq receive_purchase_request.form
                      process_purchase_order.form
                      process_purchase_requistion.form
                      complete_purchase.form))
PRECONDITIONS   (funds_available)
```

Figure 12: Purchase Task expressed in POISE Formalism.

begin are specified in the PRECONDITION clause. The usage of goals to give the system a higher flexibility was intended, but never implemented.

The sequence of constituent procedures is specified using special operators like alternation (') to indicate that only one of two procedures might occur, or optional ({ }) to specify that a procedure in the IS clause might or might not occur. This example illustrates the hierarchical character of POISE task descriptions and the decomposition into rigid subprocedures.

Experience with the POISE system showed the potential of task support offered by a system that has an internal representation of the tasks. POISE also indicated that the hierarchical decomposition of tasks is a powerful method in dealing with loosely structured tasks. Towards the end of the POISE project

it became apparent that the rigid decomposition into hard wired subprocedures was too inflexible to cope with real world environments. The designers of POISE tried to overcome this problem by replacing the hard wired decomposition of tasks with a goal-based one. This augmentation was never fully implemented. The experiences in task support gained with the POISE system led to the evolution of the POLYMER project, which is reported in more detail in the following section.

2.2.4 Plan Based Assistance

The plan based formalisms introduced in section 2.1.1 are used for a number of applications. Process control, path planning for autonomous vehicles, and plan recognition are just a few. A large application area that concerns the interaction of humans and planners is the assistance in complex or tedious situations.

In the introduction to this dissertation and in section 2.2.3, we suggested that activities are an essential part of human life, particularly of the professional part of it. We already encountered systems that offered support in the execution, planning and anticipation of activities. We also discovered the limitations of these systems, due to their rigid representational formalisms for activities.

The planning formalisms introduced in section 2.1.1 offer a powerful and flexible way to capture descriptions of activities. Based on the planning paradigm we find a number of systems that offer task support.

GRAPPLE, a successor to POISE [CL84], is a plan recognition system that offers support by observing actions of a human [HL87]. GRAPPLE tries to fit the

observed actions as subgoals into plans to infer the goal of the acting human. Knowing the top level goal of the person, GRAPPLE can offer assistance by anticipating other low level actions, inferring intermediate goals or suggesting alternative plans.

POLYMER, another successor to POISE, has been designed to support activities that are loosely structured, can have many exceptions, change frequently and involve both user actions (such as decisions or manual operations) and actions with system tools.

To address the problems that arise from these characteristics, POLYMER uses an integrated representation of activities and other objects in the environment, and provides flexible execution of activities with an interactive planner. The planner is interactive in the sense that users help with the planning decisions and force the planner to react to new situations by taking unexpected actions.

Issues studied in the POLYMER project include:

- Plan Formalisms;
- Plan/Execute Cycle;
- Multiagent Plans;
- Backtracking;
- Exception Handling.

The POLYMER architecture consists of an Object Management System (OMS) and a Task Manager. The OMS contains all the entities found in the environment being modeled. These entities are:

- objects such as written material, purchasable objects, locations, etc.

- activities, represented as plans or primitive actions
- agents who can execute these activities (either human agents or intelligent machine agents)
- scenarios, which are collections of facts that describe prototypical situations.

The OMS is object oriented in the sense that methods or operations are attached to entities and are inherited through abstraction hierarchies. The connection of activity descriptions to other entities such as forms can be regarded as an extension of the modeling approach introduced by POLYMER. The representation language for activities can be viewed as a version of the general concept introduced in section 2.1.1. Wff's are restricted to binary predicates and do not allow quantifiers. N-ary wff's can be modeled over multiple binary wff's.

Figure 13 shows an example activity from the house construction domain in the POLYMER formalism. The Task Manager supports activities that are initiated by POLYMER users. It controls the planning and execution of activities, and the associated exception handling and negotiation.

The basic interaction cycle between POLYMER and a single user consists of four steps. First, the POLYMER user, trying to conduct a task, posts a goal. Then POLYMER generates a plan that will achieve this goal. Hierarchical planning techniques, similar to those in NONLIN and SIPE, are employed. The planning process, iteratively expanding subgoals, produces a procedural net that eventually specifies a sequence of actions required to achieve the goal. These actions may either be executable by POLYMER (machine executable actions, like sending e-mail, updating databases) or by the user (agent executable actions, like signing a purchase contract or making a decision). Many of the plans in the

```
GOAL:            (in.possession ?builder.company ?new.house)
DECOMPOSITION:   (have.shell (in.possession ?builder.company ?shell)),
                 (do.electrical.work (status ?wiring complete)),
                 (do.plumbing (status ?plumbing complete))
PRECONDITION:    (exist ?customer ?new.house)
SIDE-EFFECTS:    (good ?builder.company credibility)
PLAN RATIONALE:  (enables(have.shell do.electrical.work))
                 (enables(have.shell do.plumbing))
AGENTS:          (?builder.company)
CONSTRAINTS:     (member ?builder.company (class BUILDER.COMPANY))
                 (member ?shell (class HOUSE.FRAMES))
                 (member ?wiring (class ELECTRIC.COMPONENTS))
                 (member ?plumbing (class PIPES))
                 (member ?new.house (class HOUSE))
```

Figure 13: Building a House in POLYMER Plan Formalism.

POLYMER environment are heavily dependent on constraints provided by the users at planning time. Therefore it follows that the plans POLYMER generates will generally be partial.

Third, the execution monitor selects an appropriate action from the procedural net for execution. In the machine executable case, POLYMER sends a message to the appropriate object in the OMS, in the agent executable case POLYMER interacts with the user to inform the user of the action to be taken and to learn about the results.

Finally, the actual action taken and its results are compared to the expected action. If both are congruent, POLYMER returns to step 2. Otherwise an exception handler is invoked [BC88].

2.3 Human Computer Interaction

Apart from the direct issues in knowledge acquisition and display we have to understand the view people have of their activities. Understanding that view is necessary in order to build systems that suit users in recall and display. It is here that the knowledge acquisition issue also turns into an interface problem.

We need to build interfaces to knowledge acquisition systems that encourage experts and accommodate their views. We need a theory that connects the experts' view of activities with that of the planner.

To build such a system we need an approach that takes into account the expert, the planner, the necessary tool functionality and the requirements on the interface. We try to find an approach and related technologies in interface design guidelines, cognitive engineering and usability engineering.

2.3.1 Cognitive Theories of Activity

In section 2.1.1 we outlined the elements of the planners' goal-based activity descriptions. We now turn to the human experts and look at theories that describe their expertise. We do this in order to find a theory that can relate the human view of activities to the planner's view and thereby guide the design of our knowledge acquisition system.

Cognitive psychology has long been concerned with the construction of models for human cognitive process. Three major paradigms for knowledge representation exist in cognitive psychology: Production Systems, Frame Systems and

Semantic Nets. In addition to these three frameworks, we also include the GPS model because of its importance and relatedness to our problem.

Production systems for the representation of human knowledge are collections of rules. Many theories of human task representation use rules in conjunction with a declarative knowledge base [And83,AB73,PK85]. The IF part of the productions is matched against declarative knowledge, possibly in working memory. Several strategies exist to handle conflict resolution in case more than one rule applies.

It is hard to capture goal-based activity knowledge in a rule based formalism. The goal and the preconditions can be expressed in the IF part, making this part of the rule more structured. Anderson splits the IF part of rules in this way but in an inconsistent manner [And83].

Decomposition can be accomplished by calling on primitive actions or by setting up subgoals in the THEN part. Side effects can also be specified in the THEN part. The IF part and the THEN part of these rules are propositional expressions. Actors and objects are primitives in propositions. Constraints can also be expressed through propositions.

Rules are very efficient in expressing procedural task knowledge. Matchings are done in the IF part and actions are taken in the THEN part. This makes rules an efficient and successful notation in psychology for expressing procedural knowledge. The awareness individuals have of these rules is very limited. Recent work in knowledge acquisition for expert systems shows that subjects are not able to report knowledge directly in rule form [BG86,BG87]. Production rules are assumed to be at a deep level in human cognition. Because of

these objections, the rule framework is not optimal as a basis for the elicitation of goal-based activity knowledge.

A frame is a collection of named slots. Every slot may take a value from a predefined set of values (domains). Frame based systems are collections of such frames. Schank and Abelson constructed one of the best known frame based systems for human knowledge representation [SA77]. Goals, preconditions, side effects and decomposition can easily be represented by slots of frames. Values for these slots come from the domain of actors, objects, relations and predicates.

Frames were developed to represent plots of stories and concepts. They were not intended to allow for the operational definition of plans. Frames only call on procedures to perform actions, they do not specify them operationally.

Semantic nets are the third general framework for knowledge representation developed in cognitive psychology. A semantic net consists of labeled nodes and labeled or unlabeled arcs connecting the nodes. A detailed description of semantic nets for human knowledge representation can be found in [LN77]. Semantic nets are less structured than frames and less action oriented than rules. This low grain size gives them a greater expressiveness.

Semantic nets, like frames, are a notation to represent declarative knowledge. This makes them awkward for the representation of procedural knowledge. Clearly rules are more appropriate than semantic nets or frames for the representation of procedural knowledge. The subjective psychological reality of semantic nets is as low as for the other two frameworks. The phenomena expressed in the semantic net notation are not open to introspection or recall. Subjects do not report their knowledge in the form of nodes and labeled arcs.

GPS is included in this discussion because it is one of the fundamental systems for both planning and information processing psychology [NS72] . Both areas draw on ideas and concepts developed in GPS. GPS is therefore neither just an AI planner, nor a cognitive model but something in between. This makes GPS central to our work, because we are interested in the connection between planning and the equivalent human process.

The GPS operator is used to transform states of the world. In [NS72] there is, however, no detailed structure for the operators. Goals and decomposition are not explicitly mentioned. Operators are defined by a pragmatic name, a precondition list and an effect list, e.g. *Name: Stack Two Cubes; Precondition: Cube 1 is free and cube 2 is free; Effect: Cube 1 is on top of cube 2 and cube 1 is no longer free.* Operators are selected by a higher process. This process determines operators that are applicable in a certain state of the world and then selects one of them according to the change it will cause towards the desired state of the world. The operator itself is simply an action that will then be executed.

Operators have no goals. The selection process in GPS is responsible for reducing the distance between the current state and the goal state.

Operators are either generated by a task analysis or by observing subjects acting in a problem domain. The first case, extraction from task analysis, bears no psychological relevance. The observation of subjects makes no claims about the reality the operator has to the subjects and their awareness of it.

The GOMS model [CMN83] is rooted in the GPS [NS72] tradition. GOMS itself is an acronym for the major components of the model: goals, operators, methods and selection rules. [CMN83] define goals as

> a symbolic structure that defines a state of affairs to be achieved and determines a set of possible methods by which it may be accomplished. ... The dynamic function of a goal is to provide a memory point to which the system can return on a failure or error and from which information can be obtained about what is desired, what methods are available, and what has been already tried. (p. 144)

Individuals strive to achieve goals. At every point in time an individual has a set of goals in a goalstack. Goals may be decomposed into subgoals which in turn go on the stack. To achieve a goal, a method must be applied. A method is a collection of ordered operators. Card, Moran and Newell define as follows [CMN83]:

> A method describes a procedure for accomplishing a goal. It is one of the ways in which a user stores his knowledge of a task. The description of a method is cast in a GOMS model as a conditional sequence of goals and operators , with conditional tests on the contents of the user's immediate memory and on the state of the task environment. ... Methods are learned procedures that the user already has at performance time; they are not plans that are created during a task performance. They constitute one of the two major ways in which familiarity (skill) expresses itself. The particular methods that the user builds up from prior experience, analysis and instruction reflect the detailed structure of the task environment. (p. 145)

The point that is important for our purpose is that methods are available to the user at performance time. Methods do not need to be generated when they are

needed through problem solving processes by the human. Methods can simply be called upon.

Methods are associated with goals. Methods give rise to operator sequences. An operator is the lowest level construct. Operators are

> perceptual, motor or cognitive acts, whose execution is necessary to change any aspect of the user's mental state or to effect the task environment. ... Behavior is assumed to consist of the serial execution of operators. An operator is defined by a specific effect (output) ... . (p. 144)

As GOMS is a performance model, most emphasis is put on the duration of an operator execution. The operator itself is specified by a verbal description, e.g.

> GET NEXT PAGE. Turning the manuscript page. Starts when the user's eyes begin to turn towards the manuscript; ends when the turned page falls flat. (p. 155)

The effect of the operator, though it can easily be deduced (in the case of GET NEXT PAGE it would be *see a new page*), is not explicitly specified.

If there are two methods to achieve a goal, a selection rule will be invoked that decides which method should be used. In each GOMS model there is a set of selection rules for this purpose. These rules are of the form "if such and such is true in the current task situation, then use method M".

The important difference between GOMS and GPS operators lies in their derivation. GPS isolates operators through an analysis of the domain. GOMS

could do that, too, but GOMS can also derive its operators from an analysis of protocols of subject behavior. Behavior is assumed to consist of the serial execution of operators. This makes GOMS operators psychologically more valid.

Reisner [Rei84] investigated simple BNF (Backus-Naur Forms) models of dialogue syntax. The interesting point for our topic is that she tried to show the psychological validity of formal descriptions of action languages for users of interactive systems. Reisner tried to use BNF's to predict such variables as

- ease of learning,
- remembering of particular command sequences,
- performance times,
- likelihood of errors.

This model is geared towards the keystroke level, helping system and interface designers to evaluate design decisions. The subjects' retrieval of commands is not explained in terms of information processing psychology but in stochastic ones. Reisner also makes no claims about the awareness individuals have about the psychological reality of their BNF's.

Task action grammars and set grammars are two similar formalisms designed to capture the "grammar in the head" of the user [PG86,GP84]. While this approach has been able to make good predictions about user behavior, it is not easily extendible to other tasks, and it is unclear if individuals are aware of their own grammars.

Moran introduces a Command Language Grammar (CLG) which follows the same principles as the previous grammar approaches [Mor81]. Though CLG is more sophisticated than BNF, it addresses the topic of psychological validity even less. In general, these grammar systems focused on human computer interaction and are not easily extendible to general task representation. The psychological validity of the concepts could be shown by prediction of response times and user errors, but there has been no evidence that individuals are able to report grammars. But that is our criterion, the conscious reality of the concept for the subjects.

The same criticism holds for Kieras' and Polson's approach to the formal analysis of user complexity [PK85]. They propose a two part production systems: one for the simulation of user behavior, and another one for the simulation of the machine's behavior. Interaction is formalized and can be modeled.

Newell and John worked on the recall of abbreviations for command languages [JN87]. Though this topic seems to relate computer systems to recall, we know from [BNC86] that there is a vast difference between recalling verbal and procedural material.

Norman presents a more general view of subject performed tasks, called *User Centered Systems Design (UCSD)* [ND86]. He presents a model for task performance consisting of sequences chosen from among seven stages of user activities:

1. establishing a goal;
2. forming an intention;
3. specifying an action sequence;
4. executing the actions;
5. perceiving the system state;
6. interpreting the system state;
7. evaluating the system state with respect to goals and intentions.

Bower, Black and Turner did research on the role of scripts in memory for text [BBT79]. Their findings are important for the organization of human long term memory and recall and recognition processes that occur in all types of interface problems. According to Bower, Black and Turner, scripts consist of recognizable chunks. Exceptions are better remembered than actions (von Restorff effect). An important part of their work is devoted to those parts of scripts that are left out in recall.

There is no single theory that relates the human view of activity knowledge to the planners. GOMS's application oriented character provides some of the necessary elements and frames seem structurally very close to plans. Still both theories make no assertions about the conscious human view of goal-based activities.

2.3.2 *Cognitive Engineering*

Cognitive engineering is the technical application of results and methods from cognitive science research. Cognitive science is, as [Sim81] defines, "*the domain of inquiry that seeks to understand intelligent systems and the nature of intelligence*". It embraces computer science, artificial intelligence, cognitive and information processing psychology, linguistics, philosophy and sociology. Through broad spectrum of disciplines, which focus on the same topic, intelligent phenomena, cognitive science can offer explanations and methods not available to any of its related fields.

The goal of cognitive engineering is to build systems from cognitive subsystems such that the resulting cognitive system is optimal according to some output or operation criterion. Traditionally one of these subsystems is a human user and the other subsystem is a computer [Woo86]. In our opinion, this view can be augmented to allow for groups of computers, as in distributed artificial intelligence, and for groups of humans using a computer system, as in computer supported cooperative work.

We also see two traditions in cognitive engineering: Guiding design qualitatively [HW83,Woo86] and optimizing system by tuning parameters [Fis88]. The goal of the qualitative approach was given above. Fisher gives a good definition of the goals of the later [Fis88]:

Let t be a task which operators must perform. Let O be the set of abstract objects which operators will encounter when performing task t, let p_i be the probability that the i^{th} object is encountered by the operator when performing tasks t, and let m represent the number of objects in O. Let S be a set of physical stimuli which represent the i^{th} object in set O, which vary along dimensions $d_1, d_2, ..., d_n$, and which could be seen by operators in task T. The dimensions are ones which either currently vary in the task or which one would like to vary. Sets $S_1, ..., S_m$ can be finite or infinite. Let $S = S_1 \cup S_2 ... \cup S_m$. Let M be a model which takes as input a stimulus $s_{i,j}$ in $S_i (i = 1, ..., m; j = 1, ..., n)$ and produces as output some response vector $r_{i,j} = (r_{ij1}, r_{ij2})$ were r_{ij1} is the expected time to respond and where r_{ij2} is the probability of an error. Let $f_t(r_{ij1})$ be a function which maps the mean response time r_{ij1} to the j^{th} stimulus in the i^{th} stimulus set to some cost and let $f_c(r_{ij2})$ be a function which maps the probability r_{ij2} of making an error to the j^{th} stimulus in the i^{th} stimulus set to some cost. Then, the goals of cognitive engineering are for a given task t:

- to build a model M which can be used to map every stimulus s_{ij} in S_i into the response vector r_{ij} ;
- to test the model empirically;
- to find that set of stimuli $S' = s_{1j_1}, s_{2j_2}, ..., s_{mj_m}$ in S which minimize the overall cost $\sum_{k=1}^{m} p_{kj_k}[f_t(r_{kj_k1}) + f_c(r_{kf_k2})]$ subject to various constraints on the uniformity of the stimulus construction.

We see those two approaches as consecutive stages in cognitive engineering. After a qualitative decision and design of the major components and interactions of a system, we can optimize its parameters. Though both approaches have been established for some time, there has been little success in applying the methodological rigor of the quantitative method to the qualitative one.

2.3.3 Interface Design Guidelines

In addition to theories and approaches to user modeling and interface design, we find compilations of unconnected rules, facts and findings. These guidelines are usually presented in handbooks and serve as checklists in the design of interfaces and systems [RH84,Gou89,Bro86,Mea85,Har85,Mon85].

Foley, Wallace and Chan assume that interaction techniques are at a final, lexical stage in the interaction process [FWC82]. Interface quality is to be determined according to the following three primary criteria:

1. speed.
2. accuracy.
3. pleasurability.

and the following eight secondary criteria:

1. learning time.
2. recall time.
3. short term memory load.
4. long term memory load.
5. error suspectability.
6. naturalness.
7. boundedness.

The authors argue that the proper selection of an interaction technique will optimize these factors. They continue by describing the merits of certain techniques for certain tasks and verify those statements with the results of experiments.

Gould identifies four principles of good interface design [Gou89]:

1. early and continual focus on the user.
2. integrated design.
3. early and continual testing.
4. iterative design.

An early and continual focus on the users can be achieved by visiting users and their organizations, observing and taping users, learning about their organizations, letting them think aloud, trying the users' jobs for a while, using participative design, having an expert on the system design team, task analysis, surveys, questionnaires, etc. The required steps in the design of good systems include an accurate definition of the problems, the identification of future users, learning the system constraints, and setting behavioral targets. Then a design can be made, and a prototype built and tested in a realistic scenario. This prototype will have to be optimized in a couple of iterative design cycles, preferably with participation of users.

In his classic book "Software Psychology" [Shn80], Shneiderman gives motivation for the psychological approach to computer science, presents research methodologies such as case studies, field studies, controlled experiments and their evaluations, and measurement techniques. He presents the view of pro-

gramming as a human activity and identifies certain programming tasks: learning to program, designing programs, composing them, comprehending programs, testing and debugging programs, documentation and modification.

Shneiderman also ties the concept of cognitive style into the programming activities to explain different styles of programming. He also touches upon issues such as: software evaluation, teams of programmers, database queries and languages, natural language and the design of interactive systems. He identifies the following criteria for interface design:

- simplicity.
- power.
- user satisfaction.
- reasonable cost.

To achieve these goals Shneiderman presents the following design strategy:

- collect information;
- design semantic structures;
- design syntactic structures;
- specify physical devices;
- develop software;
- devise implementation plan;
- nurture the user community;
- prepare evolutionary plan.

A whole array of authors have compiled handbooks of guidelines that are not tied into a conceptual or theoretical framework [Bro86,SM86,RH84,Tho84,WW82]. Recurring topics of these guidelines are:

- effective wording (e.g. *minimize jargon*);
- use of color (e.g. *use green for go*);
- use of graphics (e.g. *avoid graphics for exact numeral readings*);
- dialogue design (e.g. *what you see is what you've got*);
- data entry (e.g. *mandatory inputs precede optional ones*);
- control and display devices (e.g *use voice entry when hands are not free*);
- error messages and online assistance (e.g. *keep online guidance concise*);
- implementational (e.g. *separate design from implementation*);
- task analysis (e.g. *develop an explicit model of use*);
- interface styles (e.g. *avoid multiple style modes*).

Tufte explains the factors for the clear and clean presentation of data maps and how to represent quantitative information in one display so that it can be understood from many different perspectives [Tuf83]. This sort of data and information integration helps to reduce the cognitive load of users and can aid them with new angles in solving problems and perceiving trends.

Since these guidelines are singular pieces of information, they can serve only as mediators in the design of a knowledge acquisition and display system.

2.3.4 *Interface Design Philosophies*

Every designer building an interface has a certain philosophy in mind about users in mind and also about science, and the world in general. This philosophy is reflected in their interface and system design. Most of the time this philosophy is not made explicit and only recently have researchers turned to the question of what possible philosophies exist and how they might affect interface design.

The traditional scientific philosophy was founded by Descartes [AT64]. Observation, hypotheses, experiment, theory and prediction are the major components. Building systems on this philosophy assumes control, representation and prediction of certain aspects of the system and the people using it. Within this scientific philosophy there might exist different theories and models, but all are grounded in the same Descartian principles. Rubinstein and Hersh for instance summarize their design philosophy as follows [RH84]:

> The creation of computer systems involves elements of engineering, science, and art - the practical search for results, the application of theory, and the use of skill and taste. Our design philosophy is based on the view that science and art are as indispensable to design as is engineering practice. (p. 14-15)

This traditional philosophy that has worked so well (and was the only one available) in all areas of computer science is only currently being challenged [Suc85,WF86,FGHW88]. Anthropologists and sociologists, in collaboration with computer scientists, are trying to create a new paradigm based on the works of the German philosopher Martin Heidegger and anthropological methodologies

[Hei27]. This new paradigm has sometimes been called *Theory of Situated Actions* or *Practical Action.* Heidegger's basic assumption is paraphrased simply as:

> We never have all knowledge and we cannot represent it.

Heidegger sketches a new approach to deal with questions of existence, not by trying to dissect the object, but adopting the subject's view in a non-representational, rather emphatic way. Heidegger claims that this approach might take centuries to develop. As an existentialist, Heidegger beliefs that people reflect on their lives and their knowledge only in questions of life and death.

Today, Heidegger's thinking is used to justify the absence of modeling, controlled experiment and explicit methodology. Designers cooperate with users by stepping into their shoes and *being one* with the user, its organization and their problems and needs. This experience, they claim, cannot be represented or formalized but must be experienced anew every time in order to produce systems that fit the individuals.

We are dealing with the problem of knowledge acquisition and display for planners in general, and therefore we stay with the traditional, representational paradigm. Knowledge that is acquired must necessarily be represented. Not representing it would mean to lose it, which would lead the knowledge acquisition process ad absurdum.

The second issue raised by researchers in situated actions is the contextuality of actions. These researchers reject the representability of activity knowledge because of the flexibility of human responses to novel situations. We, on the

other hand, see no reason why the context could not be taken into account in the traditional, representational view of the world. It seems not necessary to change our whole paradigm to account for a small number of phenomena in a different way. The next few years will show what kind of contributions the situated action approach can make to our conduct of science and how these contributions can be employed in the design of intelligent systems and interface design.

2.3.5 Usability Engineering and Evaluation

The commercial failure of many systems and the need to move from soft to hard methods in the behavioral sciences has led to increased efforts in the development of usable systems. Usable systems are defined as systems that meet a set of operationalized behavioral goals.

This view is relatively new in computer science, where systems were usually built in the software engineering cycle. What is now called usability was to be achieved by an existing requirements analysis and a respective transformation in systems specifications. The major difference between specifications and usability however, is the focus. Requirement analysis concerns mostly technical matters such as system functionality, access time, etc. Usability, on the other hand specifies behavioral goals the system has to meet. Usability engineering is user centered.

Whiteside, Benett and Holtzblatt present a methodology of usability engineering that not only demands the definition of operationalized behavioral goals, but also provides a method to develop these goals and shows how to employ the

usability goals during the development process [WBH87]. This method employs operationalizeable goals for user system-interaction, where experiments can be conducted to evaluate the goals.

Gould also proposes a strategy to build usable systems [Gou89]. His strategy consists of early and continual focus on the intended user, integrated design with the users, early and continual testing of the system and iterative design. The steps required to achieve these criteria involve the clear definition of the problem that is to be solved by the new system, the identification of the potential user group, the setting of behavioral targets, sketching scenarios and building prototypes and the iteration of design through rapid prototyping.

Evaluation of the user system interaction is a crucial part of such an approach. Performing experiments to assess the achievement of usability goals should be no problem, given a clean operationalization of the goals.

In the case of software evaluation without having previously set usability goals, we face a different situation. Performance criteria must be considered post-facto. Methodological decisions must be made.

We can currently observe two different methodologies to evaluate software outside of the usability engineering framework. The first one is the traditional experimental approach. The performance of a treatment group of users is compared to a standard group. This is the orthodox psychology or human factors approach. This approach has been recently challenged by a new approaches called *evolutionary design* [SC88], using ethnographic methods.

Based on the principles of the situated approach discussed in section 2.3.4, evolutionary design tries to avoid preconceived notions of what might matter to

the user by starting the evaluation without any usability goals or hypotheses. The first phase of the evaluation concerns the formulation of relevant questions, which can feed back in an iterative design cycle. To achieve this the evaluators have to observe the users in their *natural habitats.* Observing user interacting with the system, or executing the users' tasks themselves, the evaluators try to identify the topics relevant to the usability of the system in the particular environment. In the next phase the evaluators have to formulate hypotheses and try to verify these through their observations in the field.

2.4 Summary of Underlying Requirements, Theories and Techniques

Analyzing the structure of plan libraries, we recognized that a knowledge acquisition system for goal-based expert systems must acquire goals, preconditions, decomposition, effects, plan rationale and constraints. Previous knowledge acquisition systems acquire either object hierarchies or rules. No knowledge acquisition system exists that acquires hierarchical plans.

Knowledge elicitation and knowledge specification require the highest degree of expert-system interaction in the knowledge acquisition process. Recall is recognized as a suitable technique for elicitation. The metaphor of paper-based forms seems adequate for knowledge specification. Previous research shows that non-programmers can interact with computers most easily in a visual programming paradigm. This is true for entering data and programs as well as for the dynamic display of results.

Assistance for users conducting complex tasks requires a goal-based representation to provide flexibility. An example of a system that offers goal-based task support is POLYMER. To satisfy the constraints of the plan libraries of such a system and to support the knowledge acquisition process, we identify the need for a theory linking these constraints to the human view of activities. In search of such a theory we turn to cognitive psychology and human-computer interaction. We discover that no such theory exists. We do recognize that the GOMS model and the theory of frames offer avenues to a solution.

The cognitive engineering approach serves as a basis for our overall methodology. We distinguish qualitative cognitive engineering from quantitative cognitive engineering. The former is concerned with high level design decisions, while the later deals with optimization issues. Guidelines help us to make design decisions in this process. We decide not to integrate situated actions for *design and* evaluation in our cognitive engineering approach because of the current lack of clear methodology in situated action research. Usability engineering serves as evaluation technique for the final system. The operationalization of usability goals gives definite criteria for the effectiveness of the system.

CHAPTER 3

RELATING THE HUMAN VIEW OF TASKS TO PLAN FORMALISMS

A correctly programmed planner with an appropriate plan library is able to accomplish some of the tasks a domain expert can perform. This fact implies an equivalence of these two information processing systems (planner and domain expert) with regard to the achieved results. It does not imply a structural equivalence and/or an equivalence of their respective cognitive processes. The human knowledge structure might differ considerably from the planner's knowledge representation formalism. This can be formalized in the following way: Let the domain expert be denoted as cognitive system DE with the knowledge structure S_{de} and cognitive processes P_{de}.

$$DE =< S_{de}, P_{de} > \tag{3.1}$$

Let the AI-planner be denoted as cognitive system AIP with the plan library S_{aip} and the reasoning processes P_{aip}.

$$AIP =< S_{aip}, P_{aip} > \tag{3.2}$$

Result equivalence can be expressed by the value of an execution function on these cognitive systems. For a finite set of tasks that can be achieved in a domain, we can state the following: Let F be a set of execution functions defined on DE

and AIP and let R be a set of accomplishable tasks then:

$$\forall\, R_i \in R\ \exists\ f_j, f_k \in F\ s.t.\ f_j(AIP) = f_k(DE) = R_i \tag{3.3}$$

(In plain words: a domain expert$< DE >$ and a planner $< AIP >$ can produce the same results R_i when given appropriate instructions $f_{i,j}$). It does not follow necessarily though, that

$$S_{de} = S_{aip}\ and/or\ P_{de} = P_{aip} \tag{3.4}$$

Most planners are built in such a way that the plan library S_{aip} must be augmented to enlarge the range of tasks the planner can accomplish. Changing the reasoning processes P_{aip}, would be the same as developing a new planner. The best domain experts can do to augment the power of a planner is to add knowledge to the plan library. The question that directly arises from this fact is: *Are domain experts with their knowledge of the domain and its procedures able to provide a planner with the necessary information for automatic planning?* Formally we can rewrite this question:

$$\exists\ \mathcal{F}\ s.t.\ \mathcal{F} : \{S_{de}, P_{de}\} \mapsto \{S_{aip}\} \tag{3.5}$$

To answer this crucial question we have to examine the components of plans that are to be mapped into, the components of the human knowledge representation of tasks, and the way this knowledge can be elicited.

We already know the quintessential elements of plan formalisms from our review in section 2.1.1. Therefore we can write:

$$\begin{aligned} S_{aip} \;=\; & < Goal, Decomposition, Precondition, Side-Effects, \\ & Plan-Rationale, Constraints > \end{aligned} \tag{3.6}$$

Turning to S_{de} we face a harder problem. In the chapter 2 we examined frameworks, theories, models and techniques from psychology, human-computer interaction and expert systems. We were looking for facts that would at least partially answer whether or not domain experts had conscious access to their knowledge that would allow them to specify plans. While there was no single theory or model that was completely satisfying, we found a number of candidates that could form a basis for further investigation:

- the GOMS model, for its application-oriented character and its goal-directed ancestry;
- recall and association techniques, for their ease of application (no knowledge engineer required);
- and electronic forms/questionnaires (instantiations of a recall technique) which have proven successful in operational specification.

In the next section we will start from these bases and develop a framework that is suitable for answering question 3.5 completely and sufficiently.

3.1 A Framework for the Recall of Subject-Performed Tasks

As we could find no theory that makes statements about structures and processes involved in the recall of complex subject performed tasks from long term memory, we started a series of interviews and experiments to gather behavioral data in order to construct a framework. Asking people how they conduct certain tasks usually leads to a description of an example. The following transcript of an interview with a secretary illustrates this:

Interviewer: How do you go about buying an item for the office?
Secretary: You mean something small like a paper holder?
Interviewer: Yes, what do you do?
Secretary: Well, first I'd have to find a catalog, an office equipment catalog, that lists the paper holder. When I found it in the catalog, I put down the vendor, the part number, the phone number of the vendor and so on ... all that stuff on the purchase order...
Interviewer: How do you continue?
Secretary: Hmm, I'd call the vendor, they mostly have an 800 number, and ask for the current price...
Interviewer: Hmhm...
Secretary: I'd put the price down on the purchase order, too ... , hmm, and then I'd mail the purchase order to the propriety department ... and I'd have to file a copy of the purchase order in our own books.

In this interview, units of grouped operations can be observed. These units fit very much what is called *Handlung* by German psychologists and philosophers. *Handlung* is a central concept for philosophers like Boesch [Boe80]. In the psychology of sport [Tho78] and the psychology of labor [Sus73] it is the basis for human knowledge and activity. The concept was recently revived in the field of action psychology [FS85,VW85,KB85]. *Handlung* bridges the gap between cognition and action.

A *Handlung* is a *conscious, goal-directed act of a human being, controlled by will, directed towards shaping reality.*[1] A *Handlung* contains operations, conducted by a human, which transform states of reality into other states, serving a certain purpose [KB72].

[1]Definition from: Der grosse Brockhaus; 17th Edition. Translation by the author.

A *Handlung* has a hierachical structure. It consists of other *Handlungen* that are less complex [VW85]. Context and opportunity are also addressed. The appropriate context for a *Handlung* is attached to it. This knowledge lets a person select a *Handlung* that fits a current intention and a current context. Expectations of the effects of *Handlungen* are also present in a person's mind.

The sequence of *Handlungen, i.e. Handlungskette* is what is recalled in the above interview. In the remainder of this dissertation we will use the English term *enactment* instead of the German *Handlung*. The reader should be aware that the word enactment is only a weak substitute for a concept as complex as *Handlung*.

The enactment is the smallest coherent unit in the description of a task. Persons describing a task do this on a level of abstraction that seems appropriate to them. The single unit on this self-perceived level of appropriateness is the enactment. Therefore enactments are readily recalled. This individually perceived appropriateness as smallest unit, varying from person to person and from task to task, makes the concept very suitable for our purpose. It also distinguishes the enactment from GPS-operators [NS72]. While operators are "the rules of the game" as found by a methodological or task analysis, enactments are the representation of the human's perception of these rules. This makes them closer to GOMS methods. Both the enactment and the GOMS/GPS operator share the property of producing new states from old ones in the problem domain, but the evidence for the entity, its appropriateness and its structure differ. GOMS methods and operators are not derived only by an analysis of the domain, like GPS-operators, but also by observation and interpretation of the subject's behavior. This makes them already more psychologically valid than

their GPS counterparts. Enactments and GOMS methods share a very close relationship. Both are sequences of operators. Both are at a unit task level of the individual. But the enactment is more than a method. The enactment also incorporates goals and preconditions. This is not the case in GOMS because there independent goals and selection rules exist. For the individual reporting its own task behavior, a process derived from self-observation, the goal has no separated existence but it is tied into the task performed. The recall of a method or an operator will also result in the recall of the goal and the precondition tied to it. Enactments model this phenomenon by representing goals, preconditions and lists of operators in one unit.

As enactments are performed with an intention to transform states of reality into other states, the effect of the operations in the operator list of an enactment is also present in the enactment. This is another difference from the GOMS method, which does not deal with the change of the current situation to a resultant situation after the application of the operators.

3.1.1 Operationalization of Enactment

As the above definition of enactments does not lend itself to immediate operationalization, we try to formalize its aspects in information processing terms [SA77]. The first aspect of an enactment is, by the above definition, the conscious goal; in our case the intention to complete a certain task. This intention is equivalent to the goal in the GOMS model or the intention in the UCSD model [CMN83,ND86]. The goal can be regarded as a slot in a larger structure. During work on one task, this goal remains the same; only situations change,

not intention. People consciously know about this goal and should be ready to report it without difficulty.

The current state of reality is captured in a second slot. We call this slot the pre-situation. It corresponds to the second aspect of the definition of enactments. During execution of the same task (same goal) the situation determines which operators are to be applied. Norman calls this the orientation in his UCSD model. Only when the task is completed or interrupted does the goal change. When people are actually performing a task, they directly perceive the pre-situation. In the case of mental simulation, or recall, the content of the working memory mirrors this state. People who are imagining working on a certain task are able to report what the current state of affairs is by simply reporting the content of their working memory.

The third aspect of the definition is the decision to generate behavior, which in turn is observable. We represent this as a third slot holding names of the operations to be generated. This is the same idea as the method definition of GOMS. Because we are not concerned with the actual execution of tasks, but only the part of them that can be reported, we are satisfied with the name the person ascribes to a certain operation. The mapping of the names to primitives of the system will be accomplished by decomposition, explained later in this chapter.

The ordering of operations, know as *decomposition ordering* or *plan-rationale* in planning, can be deduced from the order in which operations are given. The sequentiality or parallelism of these operations is a constraint property of these entities that can be recalled by humans.

Goal: Intention of the Whole Enactment
Pre-Situation: List of Properties
Operations: List of Operations
Post-Situation: List of Properties

Figure 14: Operationalization of Enactment.

The effects are also available in the recalled enactment. We call this fourth slot, describing the situation after the application of operations, the post-situation. The post-situation is described in the same terms as the pre-situation but it includes the changes caused by the operations. This definition is *similar* to the specification of GPS operators by lists of effects. The justification for this slot comes from the explanation that enactments transform states of reality into other states of reality. The difference to the GPS operators lies in the conscious reality an enactment has for a person.

The complete representation of a formal enactment is given by the structure in figure 14. We are aware that the operationalization of such a complex concept as an enactment cannot be complete. The structure we present here will certainly have to be amended.

We assume that enactments are not directly stored in human memory. Enactments are temporarily generated when humans recall tasks. Lower level cognitive structures, which themselves are not reportable, are processed and form enactments. This property of enactments leads us to the assumption that they can be sequentialized and decomposed.

Decomposition is the conscious process of recalling the operations of an enactment as enactments in their own rights. Instead of just reporting the name of an operation, subjects would report the pre-situation for this operation in general, the goal of the operation, the post-situation and other operations used, making this operation an enactment.

Sequentialization is the process of finding a sequence of enactments that leads from one state of the world (possibly the start state) to another one (possibly the end state). The post-situation of an enactment becomes the pre-situation for the next one. The goal for these enactments always remains the same: Completion of a certain task.

It is necessary to remember that the specification of subject performed tasks is not a problem solving process for the experienced subject because the subject knows the answer already. It is a process of recalling pieces of information that are relevant to planning. Therefore we are satisfied with the mere existence and description of the processes above and need not investigate their underlying mechanisms.

In addition to the recall of tasks, we assume the recallability of objects, relations and states of the world. As these entities comply to a certain degree with those items usually encountered in recall experiments, we need to make no additional assumptions.

The proposed framework of long term memory recall of subject performed tasks fulfills the requirements posited for the function $\mathcal{F}$ in equation 3.2. Let us consider every construct in S_{aip} and how our framework maps S_{de} and P_{de} into it. Goals are a part of the enactment and are recalled with it. The same is true

for preconditions. The Pre-situation of the enactment may be more general than the precondition, but the precondition is always part of it. We will, at worst, elicit redundant information in addition to the precondition. The decomposition of goals into subgoals is achieved by the decomposition of enactments in our framework. Side effects are captured in the post-situation. The ordering is the sequence in which the operations of an enactment are recalled.

3.1.2 Set of Hypotheses

With the operationalization of enactments and the proposed subprocesses, we are able to state the following hypotheses about the recall of subject performed tasks from long term memory. It is important to realize that these hypotheses are only one group of many possible descriptions for the phenomena occurring but that they seem to be a reasonable group for our purpose.

1. A general task (goal) and a specific situation (pre-situation) will result in the recall of a specific set of operations.
2. Changing the pre-situation or the general task (goal) will result in the recall of different operations.
3. A goal, a start situation and an end situation will result in the recall of a sequence of enactments. This sequence leads over intermediate situations from the start state to the end state.
4. An operation may itself be composed of enactments and those enactments are recallable (*decomposition*).
5. An alternative enactment can often be produced if the post-situation of another enactment is not achievable.
6. Operations can be distinguished from the post-situation created by these operations, by identifying properties of their physical or mental environment that have changed.

7. Temporal and logical dependencies between operations (A before B or C enables D) can be indicated.

All these hypothesis can be verified but number 6. Section 3.4 investigates this interesting problem in more detail.

3.2 Pilot Study

The experiments in this section mark the beginning of our efforts in the research of task structure and recall. They ought to be regarded as a first, cautious exploration of the problem. We started these experiments with no preconceptions about human task and activity representation. The framework in section 3.1 on page 91 is based on the interviews conducted in this pilot study and the results of the review in chapter 2. We conducted the exploration in three phases:

- unstructured interview phase to gather raw observations.
- forming hypotheses (section 3.1.2 on the basis of the unstructured interviews and the models from section 2.3.1).
- testing the hypothesis in structured problems (reported in this section).

In the first phase we asked three secretaries in the department of computer science to describe a task they perform frequently. The tasks involved forms, interaction with other people and activities of the secretary. The interviews were informal and unstructured.

From the answers and observations we formed hypotheses about the way secretaries think about their work and the way they report it. We extracted the enactments the secretaries performed in order to complete the task they described to us. This might seem odd, as enactments are by definition what subjects report. The explanation is that our theory was not formed when we conducted the exploratory interviews. We modeled the state of the world (pre-situation) after the application of each of the enactments in terms of which documents and information would be accessible to the secretary.

In the third phase, we presented each of the secretaries with very specific question, based on the previous interviews. As the interviews gave us a part of the secretaries' activity representation, we could test our theory on a very detailed level and ask very specific questions. These experiments were held six months after the interviews, to exclude ceiling effects.

3.2.1 Existence of Enactments

This experiment is intended to show that the recall of procedural tasks results in distinct enactments which are different from declarative knowledge. The following task was given to the subject:

> *You come back from a vacation and want to resume the work your temporary replacement had started in the meantime. Unfortunately you do not have a chance to talk to the person when you come back,*

but you know he was doing things exactly as you would have done them. You know he was working on purchasing an item. You find the following items and notes on your desk:

- *a note from R. requesting 10 ribbons for typewriters;*
- *a note with the item number and its price;*
- *an open supply catalog;*
- *a list of graduate students;*
- *a note with a vendor phone number;*
- *a note with object code and vendor code;*

What do you do next?

From the interviews we conducted earlier with this subject, we expected the following answer: *get a purchase order and fill it out with the information provided by the notes.* This answer, or a semantically equivalent answer would support our model because the secretary would recall the same enactment as in the interview. Now we can try to elicit the recall of this enactment, by cueing with the situation.

The answer given by the subject was:

> I check if the item has already been ordered, if a purchase order has been filled out...then I check if we have enough funds...("you

cannot find a purchase order for this process" - experimenter)...well, I have to fill one out, all the information I need is given... .

Analysis of the answer leads to the following conclusion: the situation is correctly identified, there are adjustments to ensure the validity of the data because of the artificiality of the situation (mistrust in temporary replacement), after an integrity check of the situation the expected enactment is recalled by its procedural name.

3.2.2 Decomposability of Enactments

This experiment is to test whether enactments are decomposable and if the constituent enactments are easily recallable by the subject. We picked one act the subject mentioned in the interview and asked the subject to break it down into constituent enactments. We embedded this into the experimental context in the following way:

> *For a change, let's say you get to meet the person before you leave. You are explaining to him how you work on purchasing items.*
>
> *You just explained that you file a copy of the purchase order in your own books for bookkeeping. He has never done that before. Please explain to him how you file the purchase order.*

The answer given by the subject, *when all is signed I take out the department copy of the purchase order, punch it, go to the book for the appropriate account number, look up the category (supply, ...), I enter the information to the master sheet in the right category and I put the copy right behind the master sheet into the book*, shows that the subject is easily able to decompose the given enactment. The existence of lower level enactments, serving a lower level goal which was not appropriate in the original sequence, is obvious. New objects like master sheet, local to the lower level enactment, are mentioned.

3.2.3 Difference between Post-Situation and Operations

This experiment is designed to show that the post-situation created by the operations can be recalled separately from other parts of the enactment. The post-situation is the set of expected effects and information redundant to it, but important to the person. We tried to elicit a description of the post-situation in the following way:

> *How does the action you took affect the collection of items, notes and forms that were previously on your desk?*
> *What changes?*
> *What appears and what disappears?*

The given answer,...*everything else would be removed from the desk...I'd put the catalog back...a card with the vendor information is created...I put that in a file*

listed by vendor and by item...I toss away the note to buy the item...I hold on to the purchase order until R. has signed it, confirms our hypothesis. Items are created and deleted as expected. Effects of operations are described without executing the operations themselves or simulating them mentally.

3.2.4 Goal and Pre-Situation Determine Recall

This experiment is to show that there are enactments in a distinct memory and that the recall of a specific enactment is elicited by a goal and a pre-situation. The goal and the start situation are given, and the subject is asked for the next enactment. We gave the subject the following question: *Now imagine instead of finding those items just mentioned, finding the following items instead. By now you know that the intention is to work on a proposal for a grant.*
You find the following items and notes on your desk:

- *request for a proposal*
- *a copy of a budget*
- *a supply catalog*
- *a draft of a proposal from B.*

What do you do next?
The expected answer, we deduced form the earlier interview, *either go through*

the rewrite cycle or, if everything is correct, send it out, matches the one given by the subject: *I put the budget into the draft, look at the request and see if I can make any input...I write up a new draft and give it to B.*

The subject identifies the pre-situation correctly, sets bindings and gives the two next expected enactments.

3.2.5 Evaluation of the Results of the Experiments

We only tested our theory on three subjects. The interview and the results from the experiment helped us to understand the recall of tasks in terms of planning. Our framework is based on these simple observations. Having assessed the knowledge structure of the subjects, we were able to test our framework in a detailed form.

We can summarize as follows: Subjects could recall distinct enactments, given goals and pre-situation. Subjects distinguished between operations and their effects. Enactments were successfully decomposed. Due to the small number of subjects and the highly individual experimental procedures, we dispensed with testing and statistical analysis. Three computer science secretaries are hardly a representative sample of the work force.

The scatter in the experiments is particular to the experimental setting: Good secretaries do not trust their temporary replacements. This technicality

can be avoided by a different experimental design. This is done in the next section.

3.3 Enactment Theory Experiments

This section reports a large scale experiment testing the theory. This experiment was designed to avoid the technical problems encountered in the pilot study and to provide statistically more valid data by using a larger group of subjects. Our subjects were 134 college freshmen at the University of Massachusetts. They were computer science and non-computer science majors enrolled in the computer science department and in the School of Education. Since it was impossible to interview every single one of them, as we did in the pilot study, to obtain a subset of their particular enactment memory, we decided on a task domain that would probably be known by all students. On one hand this task domain had to be restricted to only a few sequences of enactments, and on the other hand it had to allow for enough variation in the possible sequences of enactments. We eventually decided on the general tasks of "checking out a book from the main library" and "studying in the library". These general tasks involve various situations and actions, enough to allow for a number of different sequences of enactments to achieve the goal. Figure 15 shows a directed graph of enactments possible to achieve the goal "have book". Every path from the start situation to the final situation is considered a valid sequence of enactments.

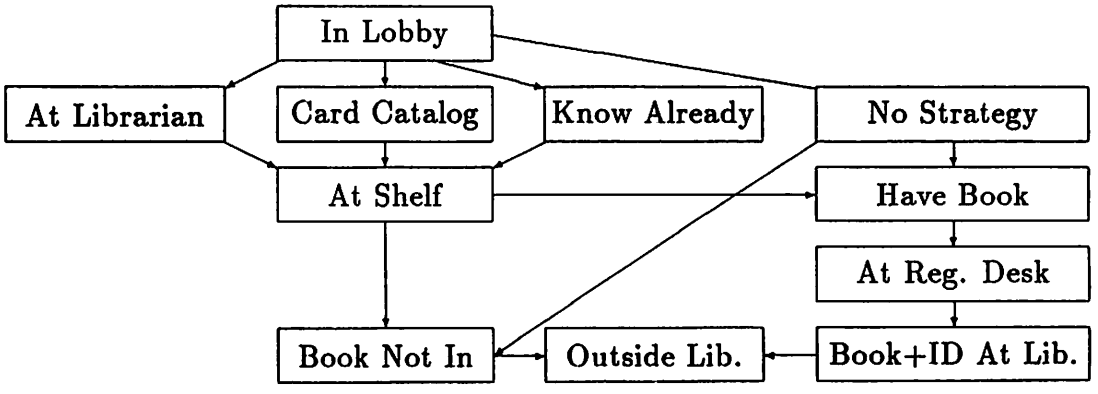


Figure 15: State-Action Graph for "checking-out-a-book"

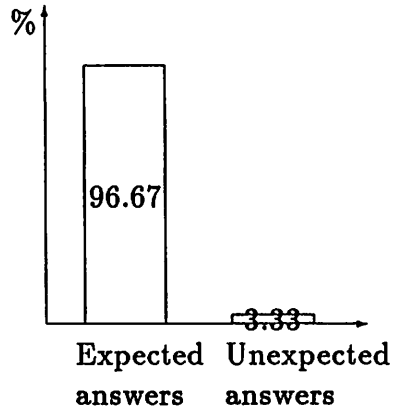
3.3.1 Goal and Pre-Situation determine Recall of Enactments

To see whether subjects would be able to recall one enactment of the set of enactments, we told them that they had just entered the library tower with the intention of checking out a certain book. This means that we put them at the top state in figure 15. We then asked them for the next thing they would do, expecting one of the of the following answers:

- ask librarian for location of book;
- go to card catalog;
- know already where book is;

- search randomly through stacks.

The answer distribution was the following:

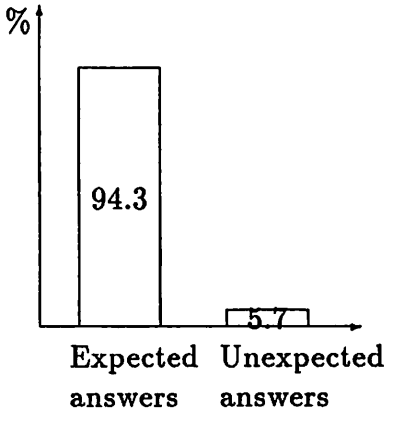


Only one out of 31 tested subjects gave an unexpected answer. Assuming that this was due to our framework rather than the experimental set up or our supposed knowledge structure about student library behavior, it still means that at least 96% of the population behave according to our framework.

In a different group of students we presented the same situation but altered the goal. Now the students were not to check out a book, but to go to the library to study. We asked them for the next thing they did once they entered the lobby. Again we devised a set of expected answers:

- take the elevator to one of the study floors;
- go down the stairs to a study area.

The answer distribution with this different goal was the following:



Two out of 35 subjects gave unexpected answers. This means that at least 94% of the tested subjects are in support of our framework, assuming no other factors biasing the results.

The variable we manipulated in the previous experiments was the goal of the enactment. One of our claims is that a different goal will result in the recall of a different action, even in the same situation. This is the major distinguishing factor between our framework and traditional *Stimulus-Reaction* theories [2] and "cognitive" *Stimulus-Reaction* theories [3]

We now wanted to test whether the manipulation of the goal from the first to the second experiment has a considerable effect on the subjects behavior. It

[2]Orthodox Stimulus-Reaction theory rejects all cognitive processes and tries to explain observable behavior (reaction) only in terms of stimuli which should preferably be controllable and measurable [Ski57].

[3]These theories try to identify cognitive processes that accomplish the connection between environmental, proprioreceptive or mental stimuli and mental or observable reactions. Most theories model those processes by IF-THEN rules. [And83].

Table 3: Precondition/Goal Experiments Summary.

	Responses fitting Goal 1	Responses fitting Goal 2	Other Responses	Number of Subjects
Goal 1 and Pre-situation 1	30	0	1	31
Goal 2 and Pre-situation 1	0	33	2	35
Number of Subjects	30	33	3	66

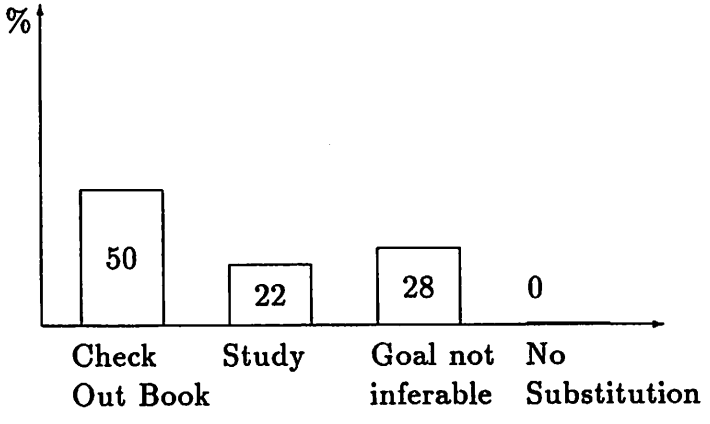
this were the case, then the recalled enactment should be completely predictable by the goal given. All subjects given one goal should recall one enactment, all subjects given another goal should recall another enactment.

The goal in the first experiment was to check out a book, in the second experiment it was to study in the library. Dropping the subjects that did not behave according to our framework in the first place, we have 30 subjects under the first goal who gave an answer from the set expected for goal "check out book" and 33 subjects who gave an answer expected for the second goal-condition "study". Table 3 summarizes these numbers and conditions. The null hypothesis would be that each cell in the table should have the same number of subjects in it as the other cells, meaning changing goals has no effect. Clearly this is not the case. We can reject the null hypothesis at a .01 level.

Another experiment concerning the effect of the goal of the general task on the enactment recalled, was conducted with a third group of students. Different groups had to be used to guarantee the independence of the experiments. In this case, the subjects were given no goal at all, just the situation. We expected that in the absence of a goal, the subjects would generate a substitute goal in order to do anything at all. We expected the following set of substitute goals:

- check out book;
- study;

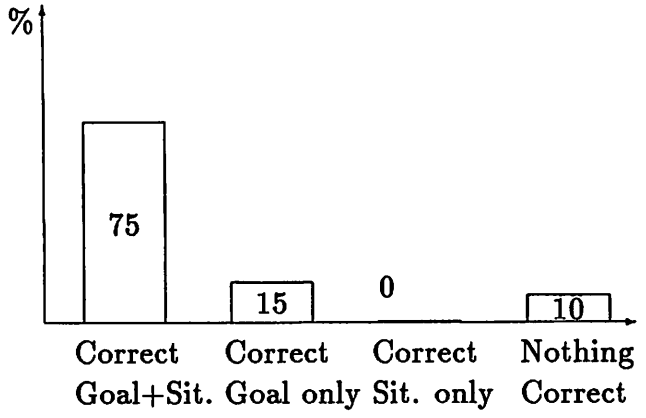
These are the only reasonable things a student could do in this library. The following diagram shows the answer distribution:



The results show that the subjects substituted goals as we expected. Those subjects whose goals were not inferable gave answers that could have either of the two aforementioned tasks (check out book, go study) as a goal. To explore this issue more deeply, we would have had to devise a different experimental setting. We did not do this, because the issue of inferring and generating goals is only of minor interest to our framework and the system we intend to build.

To finish the part of the experimental series that concerns goals, we wanted to know if subjects could do the reverse of generating actions: Recognize goals and situations by partial action sequences. We described a sequence of two actions (take spare tire out of trunk, jack up car) and asked the subjects to identify the general task (change a flat tire) and the specific situation (car jacked up,

flat still on car, spare tire in stand by position). The diagram shows how many subjects could identify the general task correctly and how many could identify the specific situation correctly:

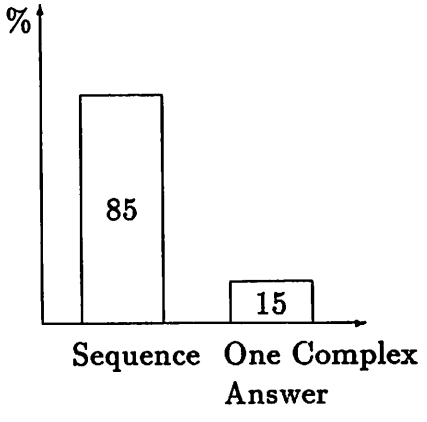


The results show that subjects can infer goals and situations.

3.3.2 Sequencing and Decomposing Enactments

The second major component of our framework is the decomposability of enactments. Subjects should be able to break an enactment into its lower level constituent enactments. This applies only to enactments subjects generated themselves, as only they know what their "standard granularity" for a certain goal is. Related to decomposability is sequencing. According to our framework, subjects should be able to start at a certain situation, mentally execute the operations of the enactment and produce a new situation. Continuing from this new situation subjects should be able to generate a sequence of enactments, all with the same goal, until the goal is achieved.

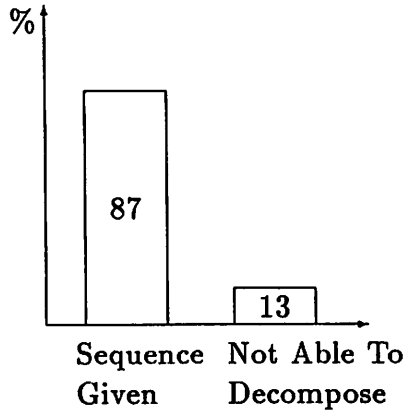
The next experiment concerned the sequencing of enactments. A goal (checking out a book from the library) and a pre-situation (found the book on the shelf) were given. Then a second situation was given (being outside the library with the book). The subjects were asked to provide the actions that led from the first to the second situation. We expected that subjects could give such a sequence that would conform with the local library procedures. The diagram shows how many subjects could give such a sequence and how many gave a complex answer being on a higher abstraction level (e.g. *just check it out*) that would bring them from the given situation immediately to the desired one:



This result did not entirely support our framework. We expected many more subjects to be able to generate a sequence than 85%. The discrepancy was not too serious though, since we still had decomposition to acquire a sequence of actions. For the plan specification system this means that sequencing and decomposition techniques have to be used side by side.

The next experiment concerned decomposition. A high level, abstract enactment has to be broken down into constituent enactments. We asked subjects

to explain how they accomplish actions they had given as answers to previous questions. We expected these explanations to be sequences of lower level actions. The subjects answered in the following way:

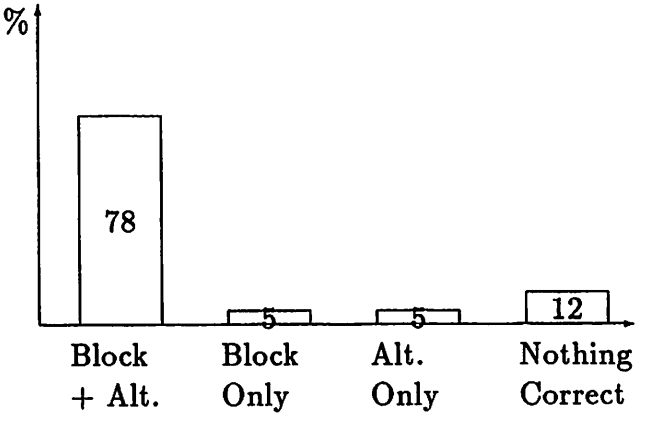


We found that 87% of the subjects were able to do the task.

3.3.3 Unanticipated Effects of Operations

The third major aspect of our framework concerned unanticipated effects. The occurrence of effects not anticipated by the subjects results in a situation different from the post-situation of the enactment. Thus the subjects had to generate an act different from the one anticipated in the post-situation as shown in the sequencing experiments. The subjects should still have been able to generate an enactment, however, because enactments are only dependent on the goal and the current pre-situation. Expectation may play a role, but our framework makes no statements about it. Eventually the act associated with the current pre-situation and the current goal should be activated.

In the experiment we first asked the subjects to give an action for a certain goal and pre-situation. Then we told them that this action was blocked. We asked if they could think of any reasons why this action could be blocked and asked for an alternative action. The diagram shows how many subjects were able to provide a reason for the block and how many gave alternative action to proceed in their tasks:

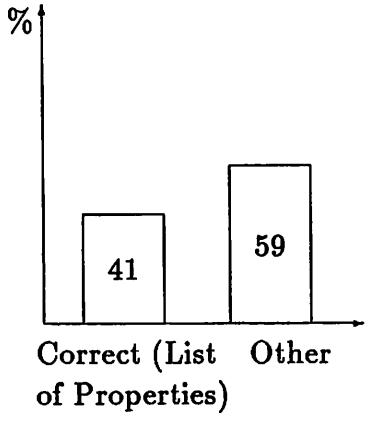


These results show that 83% of the subjects could at least provide an alternative action to proceed in the execution of their task. Even subjects who could not think of a block were able to provide an alternate action.

3.3.4 Distinguishing Post-Situations from Operations

The fourth and last major aspect of our framework is the distinguishability of operations from post-situations. We assume that subjects can easily tell what their operations result in, i.e. which effects they have. Once the subjects had generated an enactment, as an answer to a question, we asked them to give us properties of their world that had changed ("How does the action you mentioned

change the situation ?"). We counted an answer to this question as "correct" if the subjects gave a list of properties for location or possession. The following results are surprising:



These results did not fit our framework. The subjects were generally not able to distinguish between their operations and the effects of those operations. This is especially surprising since the secretaries in the pilot study had no problems identifying the effects of their operations. We will review this problem, explanations to it and a possible remedy in section 3.4.

3.3.5 Comprehensive Results

1. Goal and pre-situation determine the recalled enactment; significant at a 94% level of confidence.

2. Changing goals changes operations recalled; significant at a 99% level.

3. Changing pre-situation changes operations recalled; significant at a 99% level.

4. Enactments can be sequenced; significant at a 85% level.

5. Enactments can be decomposed; significant at a 87% level.

6. Subjects are able to report alternative enactments if the default one is blocked; significant at a 80% level of confidence.

7. Post-situation can be distinguished from operations; failed at a 41% level.

3.4 Post-Situation Reexamined

The experiments reported above suggested that our assumptions about the recall of the post-situation were not completely correct. Most subjects were not able to specify the post-situation. Subjects could not relate to the topic and did not understand our questions. To solve this problem, we started experiments exclusively concerning the effects of enactments.

We were interested in the subjects' ability to relate to the effects of an activity in terms of the primitives of a given knowledge base. We asked questions about the post-situation and the subjects had to say if those statements were true or not. For example: In the book-task in the library experiment, we asked students who said they would go down to the card catalog, if they were any longer in the lobby of the library. Students denied this. We asked if they were next to the card catalog and if they knew the books call number. All subjects were able to answer these these directly phrased questions correctly.

We concluded that the problem of specifying effects does not arise from a lack in the subject's knowledge. We detected two causes that might have been

responsible for the problem. The first cause concerns the communicational background and the level of primitives. In human communication we assume certain primitives that are operationally defined and need no further explanation, e.g. it seems strange to say: *I put the book in the drawer and now the book is in the drawer and is no longer on the table.* The sentence contains redundant information because it explains a primitive of the human language. The second cause is the way of eliciting the knowledge, the way of asking.

Considering these possible causes we designed a new experiment. This new experiment was designed to address the two cardinal questions of the specification of effects:

- can subjects specify effects operationally when prompted in an adequate way?
- do subjects recall all effects of an action?

Our subjects were thirty students enrolled in introductory computer science classes at Smith College, Massachusetts. We gave the subjects a short story, consisting of only a few sentences. The story was a semantic paraphrase of the state of the knowledge base. Then we gave a stimulus sentence that contained an action verb. We asked the subjects to specify all the resultant effects of that sentence on the story, using as many as possible words from a list given after the sentence. The list of words would correspond to the primitives of a given knowledge base of a system.
Subjects were expected to give short sentenced answers.

Although we described the state of the knowledge base in a short story and gave a list of primitives (with some distractor items), subjects were not able

to describe the post-situation. Subjects either continued the story, using the primitives given in the list, or they gave incomplete and generally useless long answers describing the goal of the activity.

We concluded that the verbal character of the story and the list of items confused the subjects. We resorted to the "form approach" which had proven advantageous in knowledge acquisition for expert systems (see section 3). Instead of telling a short story, we gave the subjects a description of the situation by means of some forms.

Then we presented the subjects with the stimulus sentence, as in the previous experiment. Subjects were given the same forms as in the description of the situation (but with no values) to fill their answers in.

The problem of specifying effects can be viewed as an instance of what is known in artificial intelligence as the *frame problem*. This problem will be encountered when an action has taken place and changed certain aspects of a knowledge base. It is not always obvious which aspects of the domain remain the same and which change their values due to the action. The subjects in our experiments had to deal with this very problem: Which aspects of the world are affected, which remain unaltered? In all three tasks subjects dealt with the frame problem in an expected way. Everything that was not explicitly mentioned in the stimulus sentence was perceived as remaining the same. Objects not mentioned, but essential to the activity, were easily recalled and updated.

The results show that we had taken the right approach to ask questions about effects. Subjects are able to specify the effects of an activity in terms of primitives of the system when given forms. Subjects are able to augment

forms to accommodate more complex situations. Objects not mentioned in the activity but essential to it are recalled and can be specified as a form.

3.5 Summary of Enactment Theory

To relate the properties of plan libraries and the knowledge acquisition process to the human experts' view of activities, we propose an enactment theory. This theory serves as a basis for the design of systems for the acquisition of plans. The theory is motivated by patterns found in exploratory interviews. Elements of cognitive psychology and action psychology are combined in the theory.

The center piece of the theory is the *enactment.* It is the product of a human experts recall of goal-based activity knowledge. We do not claim that knowledge is represented in this form, we are merely concerned with the recall of knowledge.

The properties recalled in an enactment are the *intention* of an activity, the opportunity or *situation* in which this activity is appropriate, the *steps* that are carried out to achieve the intention and the *change* that occurs in the environment due to the activity. The focus of attention during recall can change voluntarily. A step, that is part of one enactment, can be recalled in more detail as a new enactment. Enactments can be concatenated to form a chain that will achieve a complex intention. Alternative enactments can be recalled to achieve an intention if the current context excludes a certain activity. Objects that occur in enactements can be recalled seperately and be described in more detail.

Pilot studies with three secretaries and paper and pencil experiments with more than 150 students support the enactment theory. Special attention has

to be given to the recall of effects of an activity. Effects are often recalled in conjunction with the steps of an activity. Form-based recall techniques can help experts to separate effects from steps.

Chapter 4

Implications for Plan Specification and Display Systems

Our theoretical framework and the experiments show that humans, even if not familiar with computer science and knowledge engineering, can recall all elements of a plan. The answer to the question posed in equation 3.5 on page 90 is therefore "yes", humans can recall plans.

The enactment theory, the cognitive engineering approach, visualization techniques and the guidelines for interface design provide implications for the architecture, the interface and the human computer interaction of a knowledge acquisition and display system. The elicitation process, the specification process and the local debugging processes from figure 5 are primarily involved.

The implications involve the architecture of the acquisition system and the display system, the functionality of those systems and lower level interface techniques. Major implications for the architecture of the acquisition system are:

- Graphical representation of enactments;
- Graphical representation of all four enactment properties;
- Graphical representation of their values;
- Only one level of decomposition represented;
- Constraints attached to existing entities, not separate;

- Graphical representation of plan rationale by arrows;
- Consistent presentation of wff's in iconic form;
- Consistent representation of all entities in system;
- Local consistency checker;
- Debugging facilities for local inconsistencies;
- Location for caching frequently used units;
- Use of a consistent metaphor.

We can also draw a number of implications concerning the functionality of the system. These implications guide the design of the higher level tool aspects of the system:

- New subgoals start new plan;
- Various modes and patterns for display of entities;
- Granularity chosen by user;
- Labels and names chosen by user;
- Goal hierarchies determined by user;
- Enactment specification sequence determined by user;
- Careful elicitation techniques for effects;
- Querying for alternative activities with same goal;
- Emphasis of users semantic memory over syntactic memory;
- Probing memory to stimulate recall;
- Modification of existing units rather then complete creation.

On a lower level, the functionality of the interface is involved. Users are trying to manipulate entities of the interface to bring the system to a certain state or make it perform a certain function. The higher level functionality of the system could possibly be implemented by a number of interface techniques. Implications coming from interface techniques help to constrain those choices intelligently:

- Multiple interface actions to achieve effect (redundancy)
- Details of enactment only present when demanded;
- Selection of choices rather then typing of names;
- Monitoring of interface interaction, dialog navigation;
- Plan networks should be represented as graphical nodes and arrows;
- When expanding plan networks show the steps of the enactment used;

These implications, their deduction and evidence for them are discussed in the remainder of this chapter.

4.1 Architectural Implications

Our theory shows that humans see activities as a unit, the enactment. This fact must be clearly reflected in the architecture of the system. Units should exist in the architecture of the system that correspond clearly to enactments. Experience in visual systems suggests a graphical representation of this unit. A graphical editor to represent an enactment is therefore necessary.

This editor must contain all parts of the enactment. First, it must be able to represent the pre-situation and the post-situation. Then it must show the goal

and the steps to achieve it. Since we have subscribed to a graphical representation we must represent these parts graphically.

The contents of the enactment, the values of the goal, the pre-situation, etc., must also be represented graphically. Icons are the appropriate way to do that, with strings of icons representing strings of values or propositions. To be consistent, every icon must correspond to an object or activity in the expert's domain.

These icons can serve a second purpose. Users are often only interested in a reference to the object or the activity, as seen in the enactment theory. When the focus of attention shifts to that object or activity, its contents are immediately revealed. Icons can serve this purpose. They can be used to refer to units and they can be manipulated to reveal the details of the unit, possibly in a graphical editor.

The enactment theory implies that constraints are not seen as separate entities but are attached to the objects they constrain. This must be reflected in the system. Icons for units in the acquisition and display system should carry their constraints with them. The constraints are not always important. They must therefore be hidden until they are needed.

Only one level of steps is recalled in the enactment. The editor for enactments should therefore also show only one level of decomposition. Showing goal hierarchies would be inappropriate. Our experiments show that people recalling activities do not recall in terms of goal or task hierarchies but in terms of single enactments that cover the top level enactment and on a lower level the steps

needed to carry that enactment out. Experts should therefore only specify one level of decomposition at a time.

Plan rationale is easily understood when represented as arrows between steps, as our experiments show. This representation should therefore be used in the editor for the enactment. It should also be used to represent plan networks in a graphical way that is consistent with the representation of the enactment.

All entities that are recalled should have not only a corresponding editor in the acquisition and display system, but also a small visual reference. *This* small reference should facilitate the visual specification of activities. An iconic representation for all activities, objects and scenarios that appear during the recall of a task or any interaction with the system meets this requirement. Icons should show units in a comprehensive form in addition to a detailed form, such as an editor.

A specific component in the architecture of the acquisition system should verify the local consistency of the unit before it is submitted to the knowledge base. This facility would support the third stage (local verification) in our knowledge acquisition model. This component could also be used to take initiative necessary in the elicitation of effects and the elicitation of alternative activities as discussed in chapter 3.

Inconsistencies must be pointed out to the user. Again, this should be achieved in a graphical way. Highlighting of faulty parts in an editor or iconic suggestions are possible means. The users should be allowed to manipulate those very icons to correct the specified unit.

4.2 Functional Implications

Our research revealed that activities are recalled in different ways by different individuals. The cognitive organization of task domains in human cognition can differ greatly. Reorganization of the system's display of the units representing the domain is therefore mandatory. Users should be given a means to reorganize the pattern of icons representing the domain in order to approximate their individual view. They should also have means to reorganize icons in an editor for an enactment to make it more suitable to their needs and cognitive structure.

The granularity at which activities are described must be under the control of the user. Recall means that users report from their point of view and with the granularity of primitive actions they perceive. The same is true for the decomposition of activities into goal hierarchies. Users should be allowed to specify these hierarchies indirectly as they iteratively decompose enactments. No preconceived structure should be forced on the users to do this.

On a general level, users must be allowed to label entities in any way they please. This facilitates recall and guarantees relevance to the specifying expert and the advice seeking user. To create tailorable systems corporate definitions and group-specific slang must be allowed.

We learned from the experiments that the elicitation of effects can be a problem. One possible remedy is the presentation of forms for the object before and after an action is performed on it. A special facility to check for empty effects clauses after specification should be considered. This facility could presents suggestions for objects that are possibly effected in an activity.

The elicitation of alternative activities with the same goal must be initiated by the system. Our research shows that experts have a prototypical way to achieve goals but that they can recall alternative ways if asked. The best time to do this querying is when an activity is completely specified. It could be done in conjunction with the checking for effects and other consistency measures.

The decomposability of enactments can serve as a mechanism for automatic creation of new activities in the acquisition system. Specifying a step with a non-existing goal should result in the automatic creation of a new activity with that step's goal as its own goal. User should, however, not be forced to immediately attend to this unit. They must be given the choice of continuing to work on the previous enactment or doing something completely different. Dialog flexibility creates high user acceptance. To avoid confusion by underconstraining the dialog, suggestions for the most prototypical continuation of interface actions should be given.

4.3 Interface Implications

The recall of an enactment has no specific structure, just a specific result. Users should therefore be allowed to recall the properties of an enactment and specify them in any sequence they want. The structure of the dialog should allow all possible specification sequences. Suggestions can help the users to become acquainted with the system and make decisions during the dialog. It is a well-known fact that constraints can help novice users of an interface and that a large set of interface choices confuses users.

To resolve these two issues, a prototypical specification path must be spelled out. This path should act as a standard for specifying enactments. Users should be able to leave this path at all times and to return to it when desired.

The next prototypical interface actions could then be predicted and displayed in a suggestion window. This suggestion would be a continuous, non binding help message. A facility to monitor the actions of the user is therefore necessary to generate the suggestions.

The representation of plan networks was discussed above. On an interface level, the enactment model implies the presentation of iterative versions of plan networks generated in the planning process. The enactment theory states that people can shift their focus of attention from an enactment to a step in that enactment. That step becomes the new enactment and can be recalled in detail.

When the planner expands a node in a plan network, that node is replaced with the decomposition of an activity. A small subnetwork is spliced in the plan network, generating the next, more detailed version of the plan network.

This process can be mapped to the process called *decomposability* in the enactment theory. The visualization of the planning process should therefore show the original plan network, identify the node that is to be expanded, show the graphical editor of the employed enactment and replace the node in the network graphically with the steps of that enactment.

4.4 Review of Proactive Cognitive Engineering Approach

A formal method to derive design specifications from cognitive theories, such as our dedicated enactment theory, could not be found. This is surprising, since cognitive science has been applied to human computer interaction for a long time and some theories have evolved [ND86,PK85,CMN83]. Cognitive engineering and cognitive ergonomics were disciplines evolving from these efforts.

Our conception of the architectural, functional and interface implications was a result of experience in applied psychology and engineering. Existing software engineering and cognitive engineering techniques could be applied to derive and justify, but not to generate, implications. This fact points to a major problem in the field of interface design and systems design.

Cognitive theories have almost exclusively been used to analyze and explain user system interaction. As far as design goes, these efforts have resulted at best in suggestions for revisions of certain flaws in interfaces. Advice is almost always restricted to the interface, never to the system architecture. Landauer [Lan87] mentions the direct application of cognitive psychology to design as only one of four possible applications of psychology to human computer interaction. Barnard [Bar87] fails in deriving mappings from theoretical principles to system design on a general basis and speculates that the generation of systems on the basis of cognitive theories might be far in the future. Whiteside and Wixon clearly show that efforts in cognitive engineering should concentrate more on generating design specifications [WW87]. They observe that current efforts do not follow this line. Most research in cognitive engineering concentrates on the analysis of performance.

It was impossible to generate a complete methodology for the formal generation of design specifications on the basis of cognitive theories in this dissertation. In order to do that, efforts in cognitive engineering must concentrate on the proactive use of theories and join forces with software engineering.

Software engineering has long since recognized the need for requirement analysis and specification generation. Techniques to automatically generate software architectures from those specifications are currently being studied [SB89,RW89].

Information from cognitive task analysis and psychological theories must augment that process. A technique that generates design specifications based on social and cognitive requirements as well as functional and technical ones is needed.

Another approach can be taken by incorporating design and interface knowledge into the system. This has recently been considered in the area of user interface management systems and adaptable interfaces [Bra86,ML89b]. We will further explore this idea in chapter 8.

4.5 Summary of Implications

Based on the enactment theory, observations in our experiments and existing interface design guidelines, we can draw a number of implications for the design of a knowledge acquisition and display systems for plans. These implications concern the architecture of the systems, its high level functionality and low level aspects of interface interaction.

The most important architectural implications are the graphical representation of enactments, derived from the enactment theory, user interface design guidelines and experience in visual programming languages; the display of one decomposition level, derived from the enactment theory; the introduction of a local consistency checker, derived from the problems in the recall of effects; and the representation of the four properties of the enactment in a unit, derived from the structure of enactments.

Important functionality implications are the determination of goal *hierarchies* by the user, implied by the way experts shift their focus of attention; the emphasis of semantic memory over syntactic memory, suggested by the experts labeling of objects; and the choice of granularity by the user, implied by the way experts recall steps in enactments.

Implications concerning the interface are the representation of details of an enactment on demand, implied by the human focus of attention; the selection of choices, implied by the indexicality encountered in the experiments; and the representation of control flow by nodes and arrows, suggested by our experiments. All implications are obtained through inference processes. The need for stronger, more formal methods of design is recognized.

CHAPTER 5

DACRON ARCHITECTURE

In this chapter we introduce the general building blocks of the knowledge specification language and the display system. We explain their static structure and purpose in the acquisition of goal-based knowledge. The use of the system and its behavior is introduced in chapter 6.

To acquire and display knowledge we need to represent it. The basic architectural units for the representation of knowledge are:

- an underlying knowledge base, shared with POLYMER;
- a graphical representation of the knowledge base;
- objects in the knowledge base;
- activities in the knowledge base;
- scenarios in the knowledge base;
- icons corresponding to activities, objects and scenarios;

These architectural units form the basis for DACRON. They also achieve the coupling to the underlying knowledge based system.

To allow general management of the knowledge base and its graphical contents, we introduce the following units:

- a window for user guidance;
- a knowledge base navigation facility;
- a clipboard to cache icons;

To present the underlying knowledge in understandable ways to the users we need presentational forms. With the basic visualization of the underlying knowledge base and its contents, we turn to architectural units that allow visual specification of knowledge:

- graphical editors for objects;
- graphical editors for activities;
- graphical editors for scenarios
- a user action monitor;
- a reservoir for icons representing valueclasses;

A number of architectural elements must be specified to allow the display of advice generated by the planner:

- plan networks generated by the planner;
- a window for the visualization of plan networks;
- a control panel for the planning and display process;
- a window and icons to present possible variable-value bindings.

Figure 16 shows the top level block diagram of the DACRON architecture. POLYMER's knowledge base is used as the shared underlying knowledge base. DACRON creates a graphical representation of this knowledge base for the users.

Knowledge acquired through DACRON is eventually submitted to the underlying knowledge base. POLYMER uses the units in the knowledge base to conduct planning.

All units in the knowledge base are represented as icons. DACRON distinguishes between two types of units in the knowledge base. Icons for classes are presented in one window, icons for instances of these classes are presented in another window. A third permanent window shows icons representing valueclasses. Valueclasses are required to constrain the possible values of object properties in the knowledge base (strong typing).

Every icon can be opened, except for the valueclass icons. A graphical editor appears for the opened icon. DACRON contains dedicated editors for activities, editors for objects and editors for scenarios.

During the specification of new units, icons can be copied directly into editors. The activity editor has a consistency checker attached to it, that validates local consistency of newly specified activities and encourages specification of alternative plans for the same goals.

A clipboard is provided to allow the caching of icons used frequently. Icons can be found by employing the knowledge base navigation facility. The same facility allows the graphical reordering of the graphical knowledge base.

A permanently visible help window contains guidance during the use of the system. This window suggest the prototypical next action to users though it does not restrict the choice of actions.

During planning POLYMER generates plan networks. DACRON has a window to visualize these networks. To set variable bindings or select among alter-

native plans during planning DACRON presents a binding window with alternative choices. The planning and display process can be controlled by the user via a control panel of soft buttons.

5.1 Underlying Knowledge Base

The underlying knowledge base contains three basic types of units: activities, objects and scenarios. These units are represented in the POLYMER plan formalism language.

All units in the knowledge base are organized hierarchically. Units on the lower level inherit properties from units on higher levels. An object oriented data model is employed to implement the knowledge base [ZW86,ACM81].

We take a type construction approach to the specification of types. This approach includes the structural constraints, i.e. the schema, and the integrity constraints which are traditionally outside the schema. We are basically following the idea of type constructors suggested by Brodie [ACM81].

When DACRON is initialized, it creates icons for every unit in the knowledge base. Predefined bitmaps exist for scenarios, activities and objects.

DACRON contains a number of predefined top level classes of objects. These objects can be customized for any domain. Predefined bitmaps exist for the top level subclasses of the object class.

A possible set of predefined classes is Animated.Object, Agent, Human.Agent, User, People, Written.Material, Purchasable.Object, Money and Messages. We will

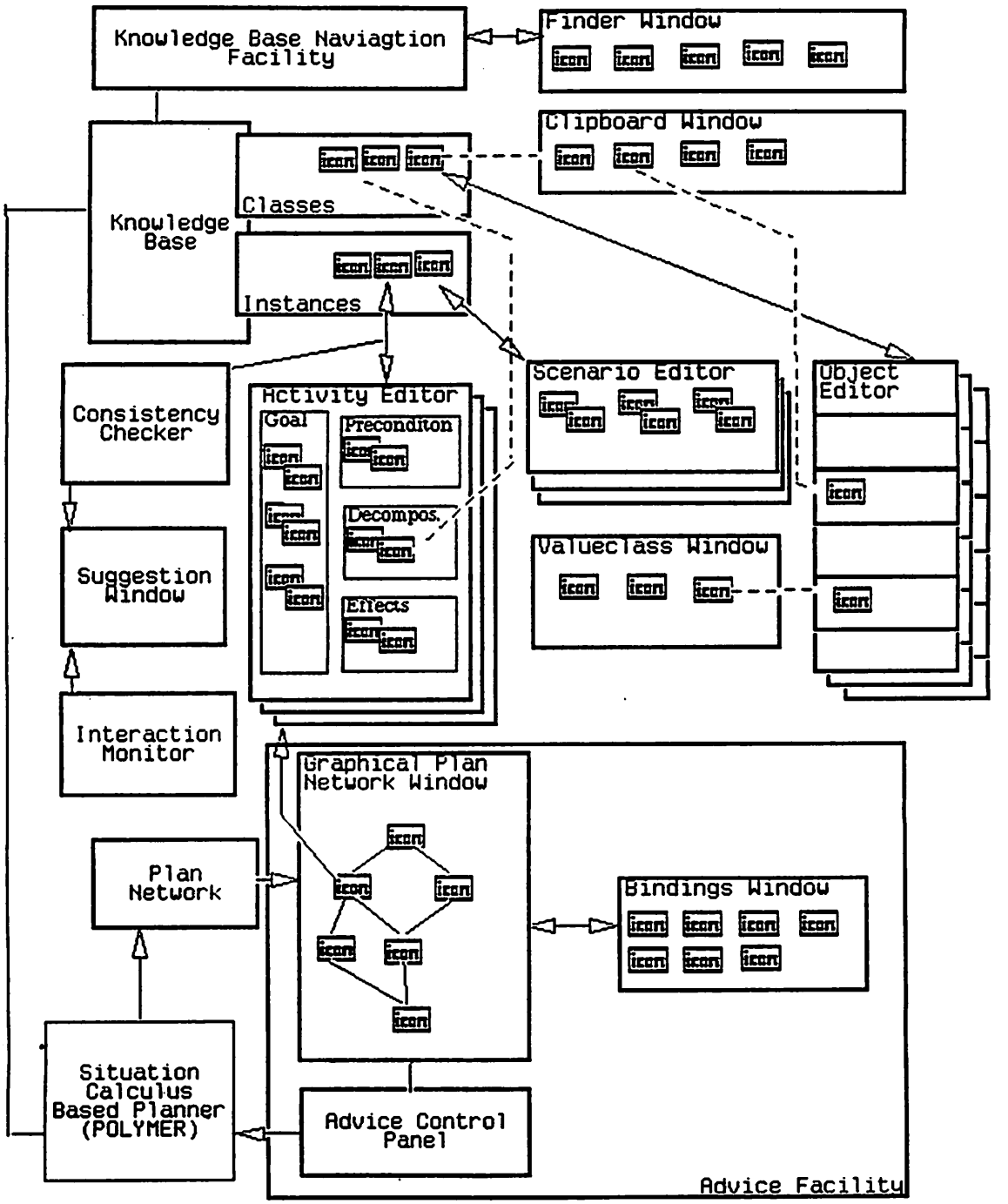


Figure 16: DACRON Architecture.

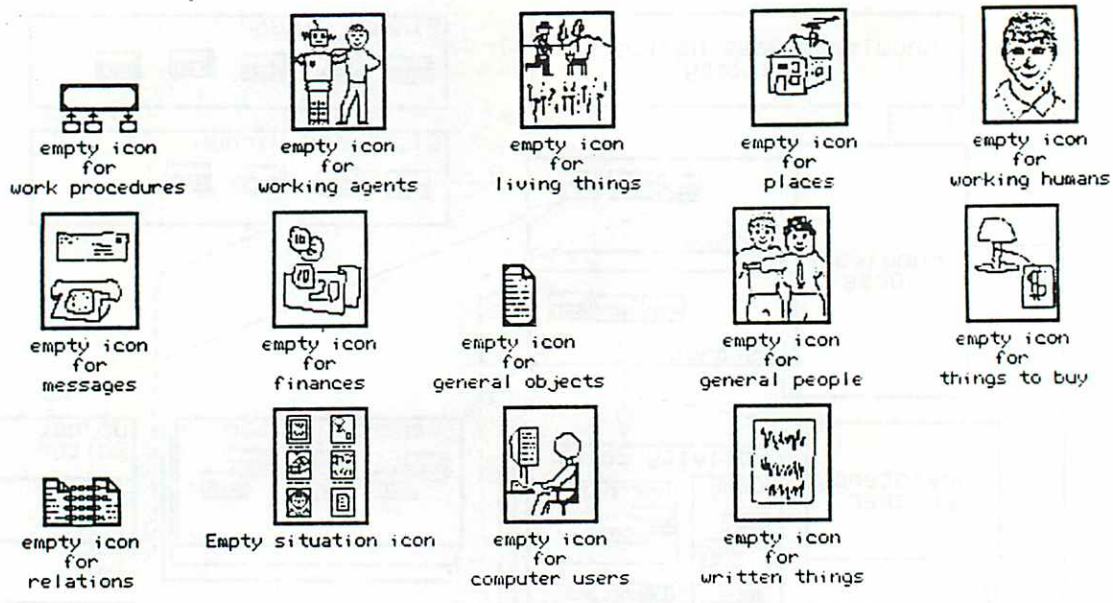


Figure 17: Predefined Bitmaps and Generic Labels.

use this set in the remainder of this dissertation. Figure 17 shows the bitmaps for the predefined classes. Depending on the specific domain, the set and the bitmaps could be replaced with more appropriate ones.

Different bitmaps for dedicated domain knowledge bases are easily created by a DACRON programmer. Every icon consists of the appropriate bitmap and a label, which is the name of the unit. If no icon exists for a certain class, the class hierarchy is searched upwards until a class with a predefined bitmap is found. That bitmap is used to present the class at hand.

Units that describe objects can either describe the generic class, House for example, or an instance of that class, My.House. To provide users with a metaphor for this distinction, we call the classes `Empty Forms` and the instances

of those classes `Filled Out Forms`. This terminology reflects the use of paper based forms in an office.

The presence of two permanent windows, the window for empty forms and the window for filled out forms, reflects this distinction. The window for empty forms also contains an icon for the activities unit and an icon for the scenario unit. This is done for consistency reasons, as will become apparent in chapter 6.

5.1.1 Objects

Objects represent static entities in the real world. These may be physical objects or concepts in an application domain, as well as their properties and relationships.

Objects in the knowledge base are represented as collections of properties, called slots. The object House for example, consists of the slots possessed.by, inhabited.by, location, price.

Every slot may have a valueclass associated with it that restricts the values that could possibly fill the slot. The price of the house, for instance, is restricted to values of type `number`. This sort of strong typing prohibits meaningless expressions, stating for example that the price of the house is `very high`. Only numbers can be accepted as valid values for the price, making *price of house is 130,000* a valid expression.

The objects themselves are organized in a hierarchy. More specific classes of objects and more general ones are defined. The class House might have the subclasses Cape.Cod, Salt.Box, Ranch.House and Bungalow. These subclasses inherit all the slots from their parent class.

Similarly there might be a super class to House. This class passes all its slots down to House. Purchasable.Object is such a super class in our example.

The actual instance of a Bungalow is distinguished from the generic class. Instances, like My.Bungalow, can not have subclasses. Figure 18 shows the bungalow class and the instance My.Bungalow.

5.1.2 Activities

Activities capture the dynamic aspects of environments. In chapter 2 and chapter 3 we reviewed a number of different formalisms to represent activities. We also discussed their advantages and disadvantages and the reasons for deciding on plans and the goal-based approach. Figure 19 shows an activity represented in the formal language.

Activities are represented as classes, not instances. Only when the planner starts to generate a specific plan network, is an instance created. Activities are structured in a hierarchy, like objects. Build.A.House is a subclass of Build.Structure and inherits goals, preconditions, constraints, etc., from it. Build.A.House has subclasses such as Build.A.Ranch.House or Build.A.Bungalow. Figure 20 shows the class hierarchy of activities in the home construction domain.

5.1.3 Scenarios

The circumstances under which a certain activity occurs can greatly influence the expansion of its subgoals and the overall success of the particular activity.

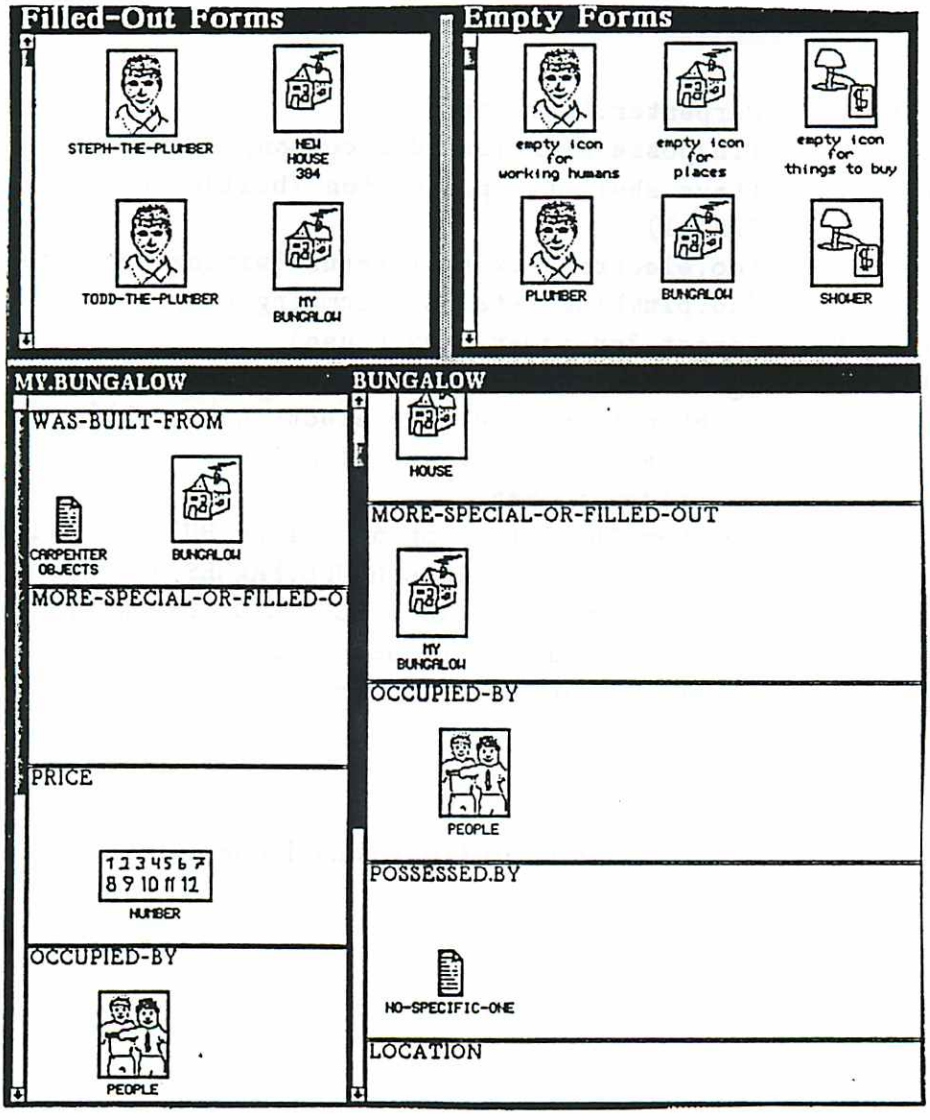


Figure 18: Instance and Class for Objects.

```
SUPERCLASS:       Carpenter.Activities
GOAL:             (in.possession ?builder.company ?new.house)
DECOMPOSITION:    (have.shell (in.possession ?builder.company
                  ?shell)),
                  (do.electrical.work (status ?wiring complete)),
                  (do.plumbing (status ?plumbing complete))
PRECONDITION:     (exist ?customer ?new.house)
SIDE-EFFECTS:     (good ?builder.company credibility)
PLAN RATIONALE:   (enables(have.shell do.electrical.work))
                  (enables(have.shell do.plumbing))
AGENTS:           (?builder.company)
CONSTRAINTS:      (member ?builder.company (class BUILDER.COMPANY))
                  (member ?shell (class HOUSE.FRAMES))
                  (member ?wiring (class ELECTRIC.COMPONENTS))
                  (member ?plumbing (class PIPES))
                  (member ?new.house (class HOUSE))
```

Figure 19: Activity in Goal-Based Formalism.

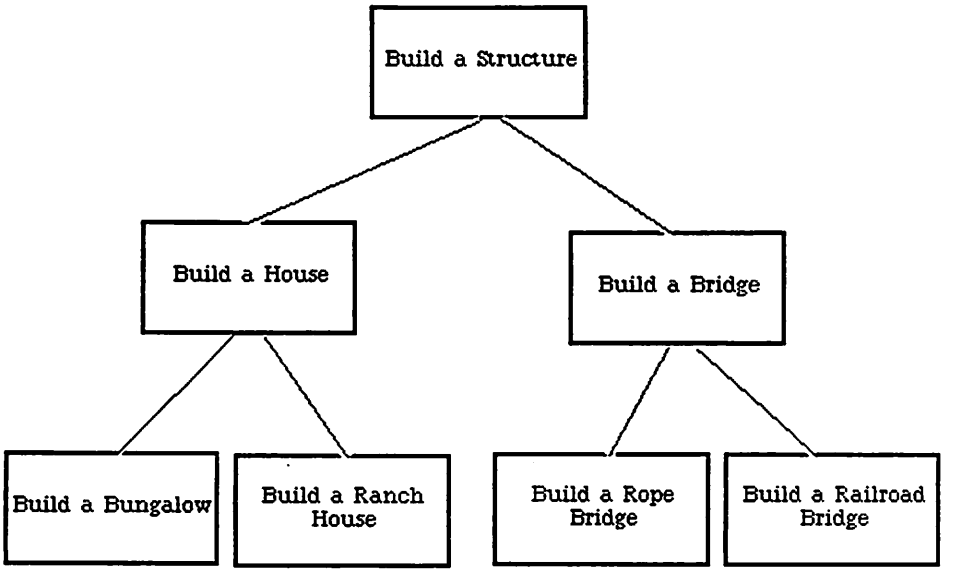


Figure 20: Activity Hierarchy in the Home Construction Domain.

The scenario units in the knowledge base provide contexts for the execution of activities. When users start an advice session with DACRON, they have to select an existing scenario for the activity.

The activity Build.A.House for instance, with the goal of having the house, must be executed differently in a congested city environment than in a rural area. While the building code in the city might limit the type of house, the ground water levels in the rural area could restrict the choice of certain basement constructions.

Scenarios are defined as collections of facts. These facts represent the relevant features of the context that surrounds the activity. Figure 21 shows the scenario for house building in a rural area as a collection of relevant facts in the POLYMER plan formalism. This scenario states that the ground water level is

```
(level ground.water high)
(area lot (acres 50))
(connected lot electricity)
(not ( connected lot sewage))
(connected lot access.road)
(exists excavating.company city)
(building.code city nil)
```

Figure 21: Scenario for Rural House Building.

high, the lot has 50 acres, it is not connected to the sewage system, an excavating company exists in the city and so on.

The creation of new scenarios allows the exploration of existing activities in new contexts. New activities can be tested in prototypical scenarios. Scenarios in conjunction with goal based activity descriptions start to provide a "situatedness" demanded by research in the area of situated actions [Suc85,WF86].

5.2 Icons

Figure 16 shows the importance and omnipresence of icons in DACRON. Though all icons consist of a bitmap and a label, they have different functionalities attached to them, depending on where the icon appears and how it was created.

5.2.1 *Proto-Icons*

Icons in the knowledge base are direct correspondents to units in the knowledge base. Every unit in the knowledge base corresponds to one and only one ***proto-icon*** in the graphical version of the knowledge base Creating a proto-icon means creating a new unit in the knowledge base. Deleting a proto-icon means deleting the unit.

Proto-icons can be copied to editors to establish references to the represented unit. The information about the bitmap and the attached unit resides in the proto-icon.

5.2.2 *Instance Icons*

References to a unit are expressed by instances of the proto-icon. These icons are called ***instance icons.*** Manipulating the instance icon does not affect the unit in the knowledge base. Only the reference to the unit is affected.

Deleting an instance icon deletes the reference to the unit, not the unit itself. Creating an instance icon creates the reference. Bitmap and label information are inherited from the proto-icon. All icons in the graphical editors are instance icons, since they represent reference to units, rather then the units themselves.

5.2.3 *Valueclass Icons*

Strong typing is achieved by restricting the possible values of a slot of an object to a certain class of values. This class of values is represented by a ***valueclass icon.*** Figure 22 shows DACRON's eleven valueclasses and their icons.

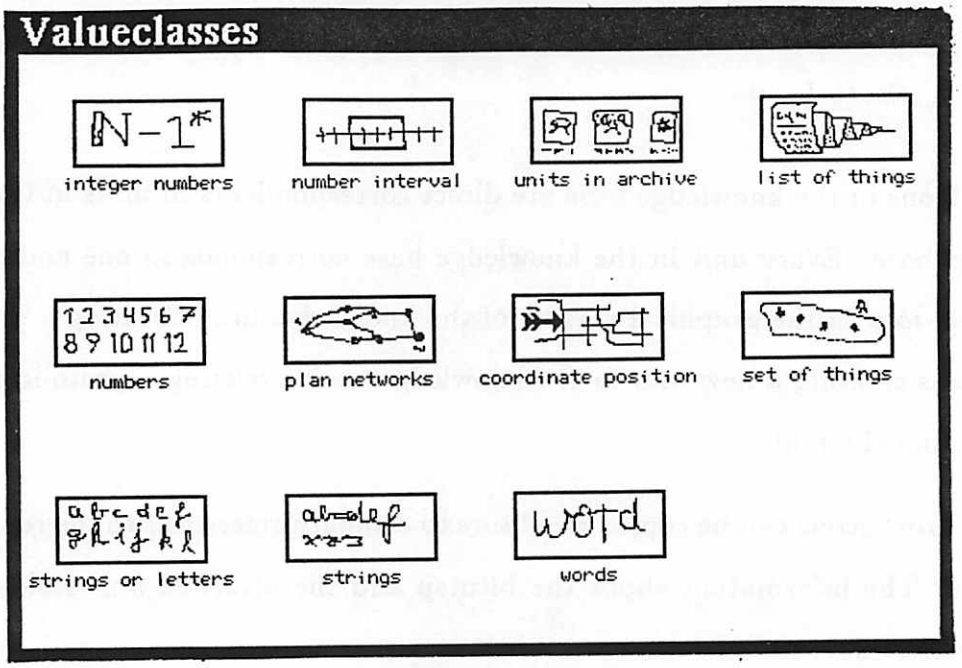


Figure 22: Valueclass Icons.

Valueclass icons are similar to proto-icons in that they carry the bitmap and the label of the icon. Instance icons can be created from valueclass icons by creating a reference to the valueclass. The valueclass icons do not represent units in the knowledge base. Valueclass icons represent sets of values.

5.2.4 Node Icons

Icons representing nodes in a plan network constitute another class of icons. These icons possess functionalities similar to those of instance icons. Node icons can be moved and opened but not copied.

Opening a node icon results in the display of an activity editor, as indicated by the arrow from a node icon in the graphical plan network window to an activity editor in figure 16. This activity editor displays the complete graphical goal the node icon represents in the network.

5.3 Editors

The content of a unit can be displayed in an editor window for that type of unit. There are dedicated editors for activities, objects and scenarios. The number of editor windows is unlimited. New windows for editors are created next to the most recent window, partly overlapping it. The activity editors and object editors in figure 16 indicate this fact by being stacked.

The number of editors associated with one proto-icon is limited to one. This restriction prohibits versioning problems and ambiguities. The design of the editors is a direct result to the interface metaphor using paper based forms and the implications from chapter 4.

5.3.1 Activity Editor

The editor for activities results directly from the cognitive framework. The editor is presented in a window that can be moved. Scrolling or reshaping that window is impossible, as it would destroy the appropriately designed structure of the editor.

The top bar of the window contains the name of the activity. Four compartments represent the goal, the precondition, the decomposition and the effects of a plan. They are labeled `Goals`, `Situation Before Actions Start`, `Actions That Are Carried Out` and `Situation After Actions Are Completed`, respectively Each of those compartments appears as a rectangle with the name of the compartment on top.

The goal compartment is a vertical rectangle that fills the left third of the editor. In this way the goal is distinguished from the temporal sequence visualized by the compartments shown on the right side of the editor.

The remainder of the editor is split into three equal rectangles. The top rectangle is the precondition compartment, representing the situation before the execution of the activity. The second rectangle is the decomposition compartment, showing the steps that are taken in this activity and the bottom one shows the effects, the situation after the steps are taken. Figure 23 shows an activity editor from the house construction domain.

Each compartment represents the corresponding slot of the underlying plan formalism. The values of those slots are expressions in predicate calculus, wff's. Section 2.1.1 defines wff's and explains their expressive power for planning. The following discussion restricts itself to wff's employing binary predicates without quantifiers. Following the pattern given below, our representation of wff's could easily be extended to handle n-ary predicates and quantifiers.

We represent wff's iconically in the compartments of the editor. Usually just the icon for the subject of the wff is visible. The wff (`has John House`) is marked by the icon for John. This saves space and allows the user to work with the wffs of current interest. The subject icon can be expanded into the whole iconic wff.

The wff is displayed by a subject predicate holder, a subject icon, a top predicate holder, an object predicate holder and the object icon. For example, the fact that the status of the tar top for the access road to the new lot is not completed is expressed by a subject icon for the access road located in a

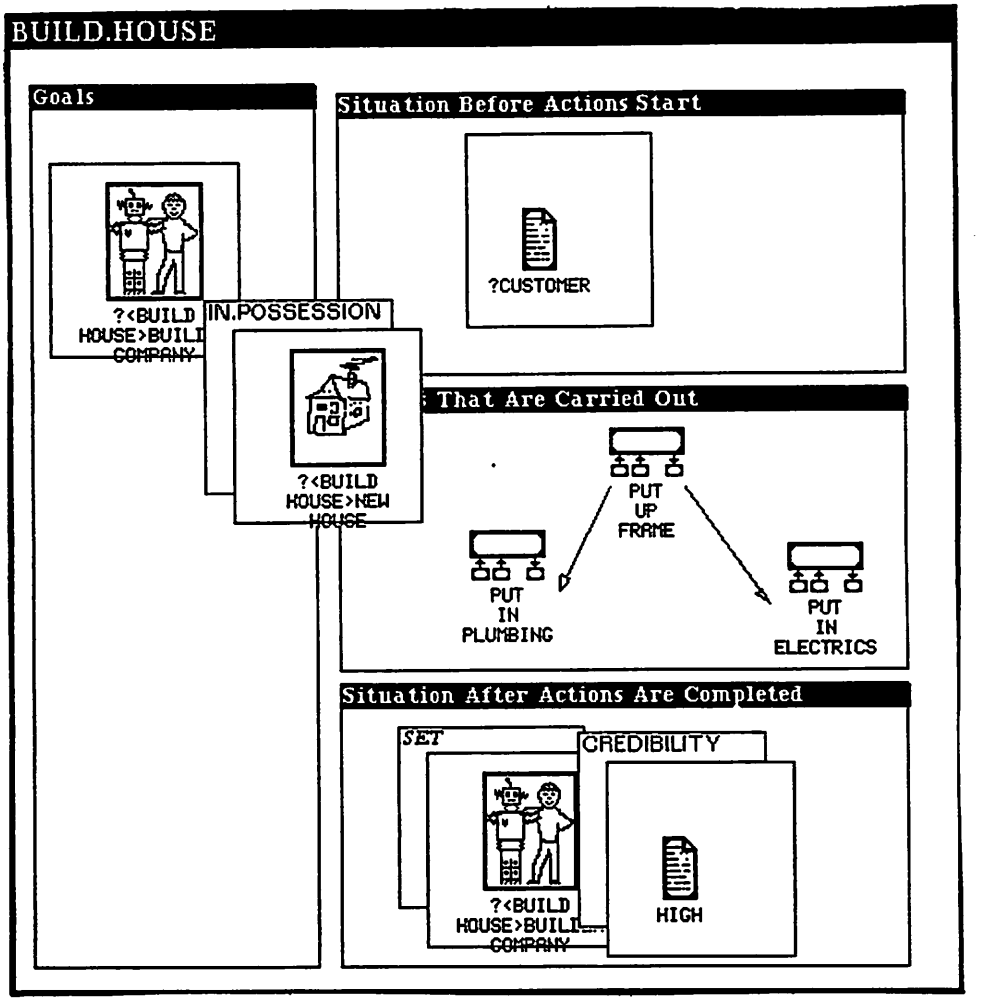


Figure 23: Activity Editor from House Construction Domain.

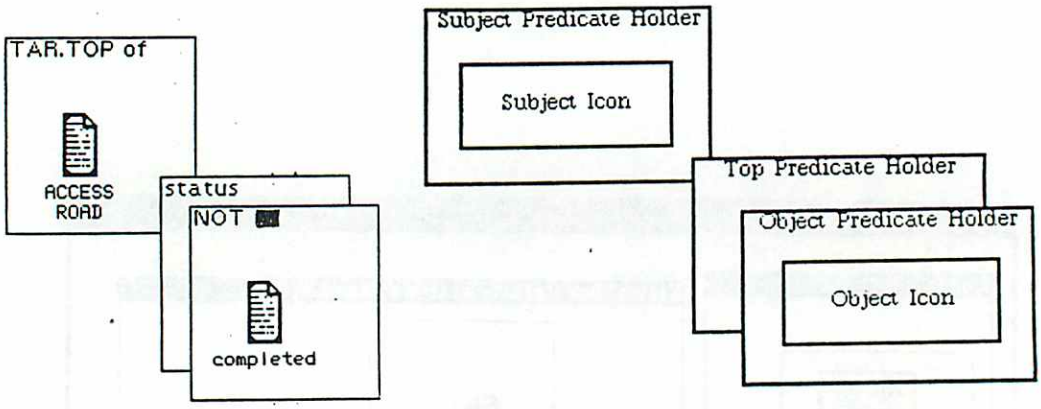


Figure 24: Hierarchy of Holders and Icons in Iconic WFF.

subject predicate holder that also contains the string *tar top of* to represent the predicate on the subject.

A top predicate holder is located on the lower right hand corner of the subject icon. In this example the string in the top predicate holder reads *status*. The remainder of the fact (*is not completed*) is represented by the object predicate holder, containing the string *not* and an icon for the string *completed*. Figure 24 shows the iconic wff on the left and the hierarchy of icons and holder on the right.

Constraints are also expressed as wff's. No dedicated compartment for constraints exists in the graphical activity editor. The cognitive framework implies that constraints are tied to particular entities already present in the description of the activity. These entities are represented as subject or object icons in the activity editor. These icons can be expanded to show constraints in an iconic wff. Figure 25 shows an example, where the icon for access road is used to show constraints on that road.

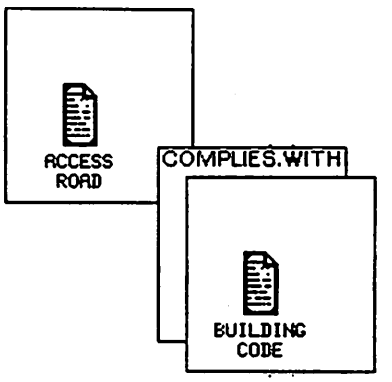


Figure 25: Constraint as Iconic WFF.

The plan rationale is indicated by arrows between the icons representing subgoals in the decomposition compartment. An arrow from icon A to icon B indicates the A has to happen before B. No arrow designates the absence of sequential processing requirements. Such subgoals can be achieved concurrently.

5.3.2 Object Editor

The contents of objects are represented in a dedicated object editor. This editor complies with the metaphor to represent all entities on the interface such as paper based forms.

The object editor consists of a sequential collection of rows. Each row represents an object's slot. Two additional rows at the top of the editor show the position of this object in the hierarchy of objects. Figure 26 shows an editor for the class House, with the name of the object in the title of the editor.

Each object editor is presented in a window of predetermined size. Usual window operations, like reshaping, scrolling and moving, can be applied to the object editor window.

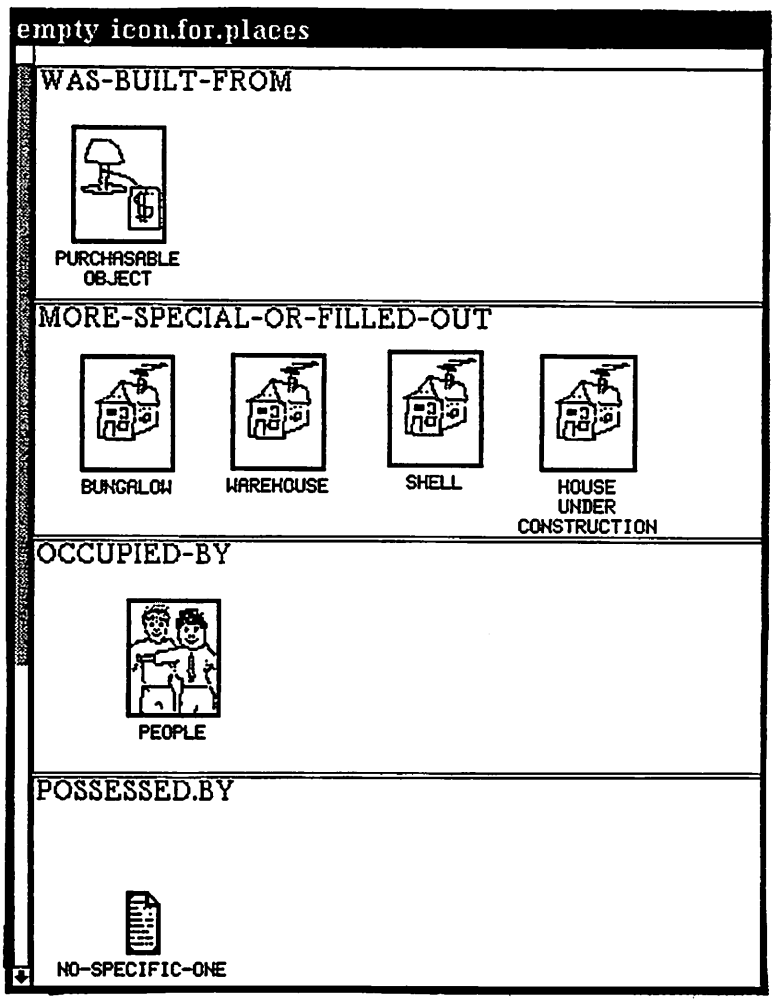


Figure 26: Object Editor.

Each row has a label in the top left corner, displaying the name of the slot. The editor is constructed as a concatenation of rectangular, labeled rows.

The first row in the editor, called `was-built-from`, shows the parent objects. All slots from these parent objects are inherited by the currently displayed object. The parent objects are represented in their iconic form.

The second row, called `more-special-or-filled-out`, shows the icons for the subclasses that were built from the current object or the instances that were created from this object.

The other rows, representing the slots, contain an icon for that valueclass that is associated with that slot. The row `occupied.by` in figure 26 is restricted to values that are of the class People, indicated by the people icon.

The value of the row is displayed in a value holder that appears only on request. This value holder is a subpicture of the valueclass icon. Inside the valueholder resides the icon representing the value. The value of the `occupied.by` row for example, is John Miller, expressed by the John Miller Icon in the valueholder. Values that are not units, are represented by default by the object icon with the value itself being the label of the icon.

5.3.3 Scenario Editor

The editor for scenarios is similar to the editor for activities. Both allow the display and specification of iconic wff's. The scenario editor resembles one compartment of an activity editor with two rows for the hierarchy information above it (see figure 27). These two rows serve the same purpose as they do in the case of the object editor.

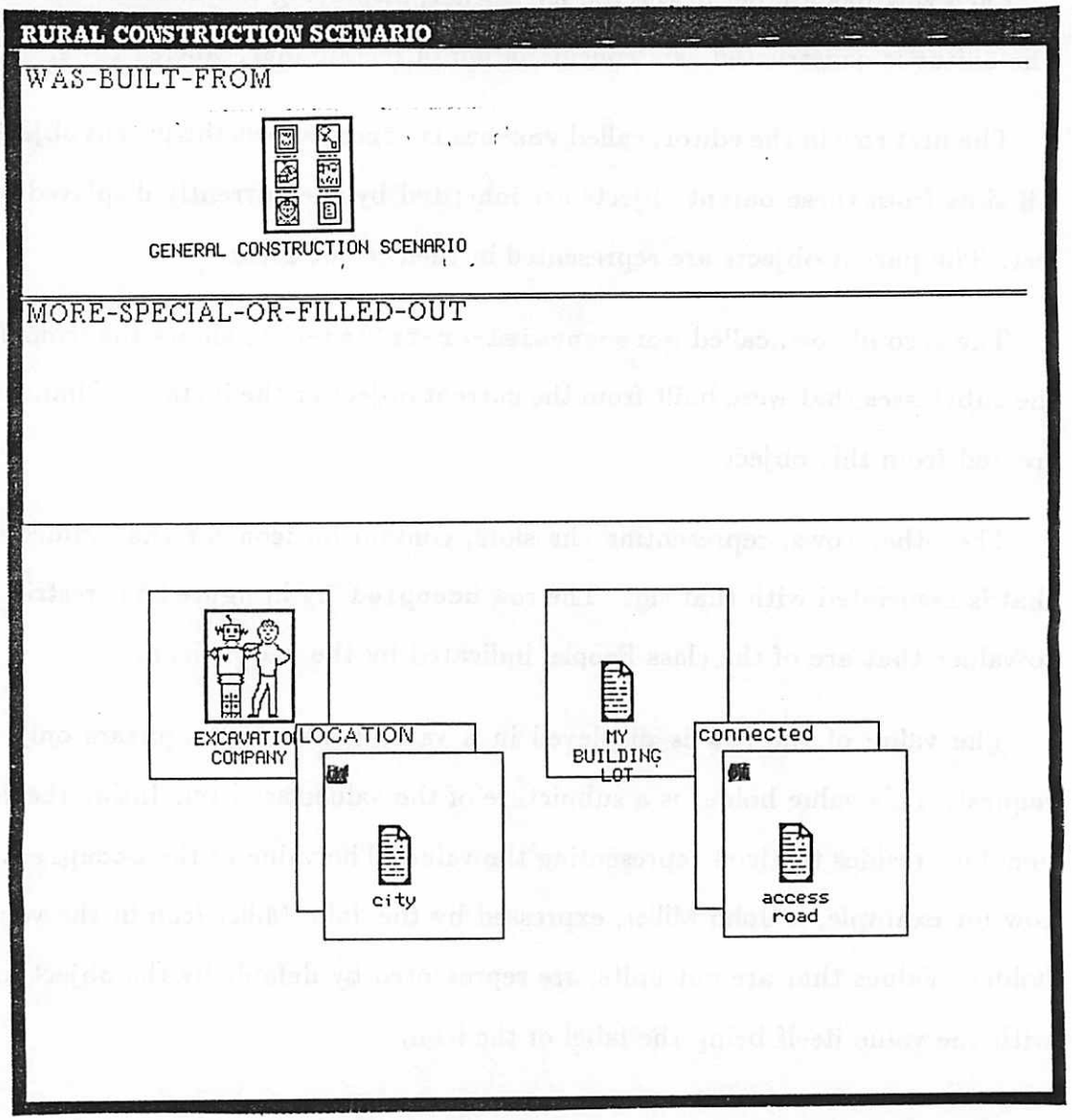


Figure 27: Scenario Editor from the Construction Domain.

The first row shows more general scenarios. The second row shows specializations of this scenario.

The compartment for the collection of facts displays iconic wff's in the same way as the compartments in the activity editor do. The subject icon is always present. On user demand the whole wff can be displayed. The display of iconic wff's is similar to the one used in the activity editor.

The scenario is the union of the facts represented by the icons. Future versions of DACRON might contain icons for set operators that allow intersection and exclusion of certain sets of facts in one scenario description.

5.4 Consistency Checker

The consistency checker validates the correctness of newly specified activities. Before a new activity unit can be submitted to the knowledge base, indicated by the arrow from the activity editor to the knowledge base in figure 16, its local correctness and validity have to be proven. The consistency checker acts as a safeguard against locally inconsistent new units entering the knowledge base and possibly disabling it.

The consistency checker can detect the following local inconsistencies:

- Circularity in the decomposition: It is meaningless for a set of subgoals to depend on each other in a circular way, e.g. (enables A B), (enables B C) and (enables C A). Such an expression can not be satisfied by a planner.
- Contradiction in precondition: An expression and its direct negation are meaningless and can not be satisfied at the same time. Such a precondition would render the activity inapplicable in all cases. It is meaningless to state that `(status house completed)` and at the same time `(not (status house completed))`.

- Contradiction in the goal: This case is similar to the precondition.
- Contradiction in the effects: This case is similar to the precondition.
- Variable reference in effects: The variables used in expression in the effects clause must be introduced in at least one other place in the activity. During run time variables are bound to values. This binding happens during precondition checking and goal expansion. Variables in the effect clause must therefore be tied to those other variables.

The consistency checker also tries to detect incomplete specifications of activities. Queries about unaddressed slots are generated to the user. The clauses the consistency checker tests for emptiness are:

- Goal: An activity can never have an empty goal clause.
- Decomposition: The decomposition can only be left empty when other experts are expected to continue specifying this activity or when this activity is intended to be directly executed by a person.
- Precondition: The precondition could be empty but experts are encouraged to be as specific as possible.
- Plan Rationale: Same as precondition.
- Effects: Users often forget to specify effects. An empty effects clause triggers the presentation of all units used in the plan and inquires about the effect of the activity on them.

In addition to checking the absence of contradictions and the completeness of the new activity, the consistency checker initiates the elicitation of alternative activities that would satisfy the same goal. Multiple activities satisfying the same goal but with different preconditions and different courses of action can be acquired in that way. The activity Erect.A.Frame.1 with the goal to have a frame for a house erected could be done by posting beams and attaching sheetrock,

this goal could also be achieved by building a cast and pouring concrete resulting in Erect.A.Frame.2.

5.5 User Interaction Monitor and Suggestions

Crucial user actions, other than submitting a newly specified activity to the knowledge base, are observed by the *User Interaction Monitor.* This monitor follows the interface actions of the users and creates suggestions based on the current state of the interface.

These suggestions are based on a prototypical session. When a user starts to specify a wff, for instance, by copying an icon into a compartment, it is reasonable to continue by next selecting the property of that unit that should appear in the wff. A message suggesting that is created in the *Suggestions Window.*

A suggestion is not a command. Users can take other actions, start the specification of another wff, or a completely new unit, for instance. The suggestion only mentions the action that would follow in most cases.

The appearance of a suggestion does not require any acknowledgment from the users. The users can ignore the suggestions completely by simply ignoring the suggestions window.

5.6 Knowledge Base Navigation Facility

To navigate large knowledge bases successfully, tools to change the appearance of the knowledge base and to locate desired items or classes of items must be provided. These tools are assembled in the *knowledge base maintenance facility.*

5.6.1 Icon Configuration in Knowledge Base

The arrangement of icons in the knowledge base windows can be done according to the following criteria:

- bitmap
- alphabetical name
- class structure

Configuring by bitmap results in all icons that share the same bitmap being arranged in the same column. Once one column is complete the next column of icons of a bitmap type is started until all icons are sorted into columns.

Reordering by alphabetical name results in an alphabetical ordering of the icons, line by line. The bitmap is not taken into account, only the name of the unit.

The class hierarchy of the units in the knowledge base can also be visualized. A class forest, multiple trees, represents the class to subclass relations.

5.6.2 Locating Icons in the Knowledge Base

Reordering the knowledge base and scanning it might not result in locating a desired icon or group of icons. The *finder* allows the user to specify queries and displays results in the *finder window.* From here the icons can be copied directly to the desired editor, or they can be saved on the clipboard.

The simplest query consists of the specification of the name of a desired unit. A string match between the query and the names of the units in the knowledge base is performed, and the result, possibly nothing, is returned in the finder window.

Units can also be located by specifying a query about properties of the unit. The superclass and subclass links are properties all units share. Other properties are specific to objects, activities or scenarios.

Activities can be queried for by goals, preconditions, effects and decomposition or the occurrence of variables. Objects can be queried for by the slot names, valueclasses used or values present in the objects slots. Scenarios can be found via the variables used as part of their fact specification of by the presence of certain facts.

5.7 Clipboard

Icons that have been successfully located in the knowledge base might be copied to the clipboard, which is a permanent window. Reordering the knowledge base or scrolling the knowledge base window do not have to result in a new search for the same icon. Frequently used icons can be cached in the clipboard.

Icons in the clipboard are implemented as instance icons. Deleting an icon in the clipboard only deletes it from the window, not from the knowledge base. Icons from the clipboard can be copied to editors or opened to change the specification of a unit, just like proto-icons.

5.8 Advice Facility

The advice facility consists of a number of modules that allows users to solicit advice on how to do certain task, how to achieve goals or to explore domains. The planner (POLYMER) is used to generate knowledge based advice on the basis of the knowledge stored in the knowledge base.

The planner and the whole advice session are controlled via the *Advice Control Panel.* Advice in the form of plan networks is displayed in the *Graphical Plan Network Window.* Whenever there is a choice from a set of values for vari-

able or choice from a set of activities to satisfy a goal, the *Bindings Window* displays these choices.

5.8.1 Plan-Networks

The result of the planner trying to satisfy a certain goal is a plan network. In its most detailed version this plan network is a complete set of actions that are to be taken by humans and machines in order to achieve the desired goal. These actions are arranged in a partial order in the plan network.

Before the complete plan network can be constructed the planner has to go through a series of iteratively refined versions of the plan network. The initial plan network consists only of a start node, a node representing the desired top level goal and an end node.

The planner attempts to refine this plan network by searching for an activity in the knowledge base. This activity must have a goal that matches the desired goal and preconditions that are satisfied by facts in the selected scenario.

When such an activity has been found, the node for the goal is replaced by nodes representing the subgoals in the decomposition of the selected activity. These subgoals can either be goals themselves in which case this process repeats, or they can be primitive actions that can actually be executed by humans or machines. In that case no further planning is necessary.

The planner keeps expanding nodes representing goals until no more nodes for goals exist, meaning that all nodes represent primitive actions, or no activity can be found. In the latter case a partial plan network is the end product.

5.8.1.1 Goal Nodes

Goal nodes represent goals that are still to be achieved. Goal nodes are represented by the activity bitmap. The label is derived from the goal represented by the node.

The contents of a goal node can be viewed in an activity editor. The goal is shown in the goal compartment. The link between the goal node icon in figure 16 and the stack of activity editors indicates this possibility.

5.8.1.2 Agent Nodes

A primitive action might be something a person can execute, like signing a contract or buying supplies. Such actions that are executable by a human agent are represented by *agent nodes* in the plan network. Since these actions are primitive, no information other than their semantic label can be provided.

At the time an agent node must be executed, a message is displayed to the human agent. The person can reply to this message in a number of ways, confirming execution, pausing it or doing an unexpected action. The later case

creates an exception to what the planner expected. The SPANDEX module of POLYMER addresses this problem [BC88].

5.8.1.3 Tool Nodes

Another type of action that is directly executable is the function performed by a machine. Nodes representing the action of a machine, computer, intelligent machine agent or any other tool are called *tool nodes.*

Because tool nodes represent primitive actions, they can not be explained further or displayed. Executing a tool node means executing a piece of code. Tool nodes are represented by activity bitmaps, labeled with the name of the tool action.

5.8.2 Graphical Plan-Networks

The programmatic plan networks are represented graphically in the *Plan Networks Window.* All nodes are represented as icons. The functionality of these icons is described above.

The icons are connected by directed arrows. The endpoints of the arrows are defined by their respective icons. Moving an icon results also in moving the endpoint of the arrow.

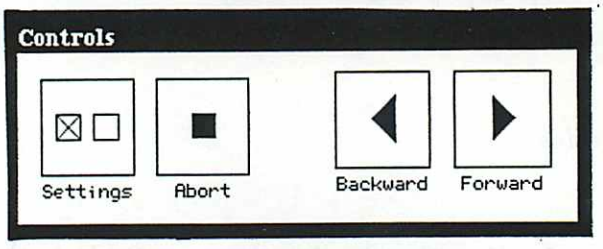


Figure 28: Advice Control Panel.

Two classes of icons are highlighted in every plan network. First, the goal node that is to be expanded is displayed in reverse video. Second, icons for nodes that were created as a result of a goal expansion are highlighted in light gray, so users can easily detect which goal node resulted in which new nodes.

5.8.3 Advice Control Panel

This panel, shown in figure 28, consists of four soft buttons modeled after common VCR controls. Two buttons, `FORWARD`, `BACKWARD`, control the plan networks. `ABORT`, aborts the whole planning process, while `SETTING` accesses a number of display parameters.

The forward button initiates the generation of the next plan network. The backward button replaces the display of the current plan network with the previous network.

The setting button allows the users to access a menu that controls the following display parameters:

- User choice of variable bindings versus random choice by planner;
- User choice among alternative plans for goal versus random choice by planner;
- Horizontal versus vertical display of networks.

5.9 Architecture Summary

In chapter 5 we turn the implications drawn from the enactment theory into system specifications. The specifications concern the representation of knowledge, the architecture of the knowledge base and its components, and the control and support of the specification and advice process. Among the major implications are the graphical representation of the knowledge base, dedicated editors for activities and objects and the visual generation and control of advice.

Based on these specifications we propose an architecture for a direct knowledge acquisition and display system, called DACRON (Direct Acquisition of ConstRained Object Notation). DACRON is based on a forms and VCR metaphor. It shares the application knowledge base with the underlying expert system. DACRON represents the knowledge base graphically in two permanent windows. One window represents classes, called empty forms in the metaphor,

the other window presents instances of the classes, called filled out forms. Every object in these windows is represented by an icon.

The icons can be opened and dedicated editors for the type of icon appear. Values in the editor, which also carry the form metaphor, are represented by icons or strings of icons. New units in the knowledge base can be created by opening editors and copying existing icons. A consistency checker verifies new activities before they can be submitted to the knowledge base. The knowledge base navigation facility allows to search for icons by a number of semantic and syntactic keys. The facility also allows to arrange icons in the knowledge base window in different ways. Advice is displayed by creating a graphical version from a plan network generated by the underlying planner. The graphical plan network contains icons already known form the knowledge base. Users can interact with the planner through a VCR-like control panel, allowing them to see more detailed versions of plan networks or to step back and review their development.

CHAPTER 6

USER-DACRON INTERACTION

This chapter describes the use of DACRON for the specification of knowledge and the solicitation of advice [1]. The description focuses on user interaction at an interface level but the results of these interactions and the behavior of DACRON are reported in terms of the underlying architecture as introduced in chapter 5.

The simplest interaction is browsing the knowledge base windows. Scrolling, looking and reordering familiarize the users with DACRON and the domain knowledge base. On the next level of complexity users search for specific units in the knowledge base. Facilities to search by different keys and properties of the units are introduced. Once units of interest are found, either by scrolling or by searching, they can be viewed in dedicated editors.

Domain experts specify knowledge by finding the generic icon for the unit to be specified, opening a new editor for it and typing the new unit's name in the editor's title. Icons from the knowledge base can then be copied to specify values.

Novices to a domain can use the acquired knowledge to solicit advice from DACRON. Advice on a task is solicited by locating the corresponding icon and

[1] [ML89a] is a videotape that illustrates DACRON's functionality and use.

starting the advice session from it. During the advice session variables in plans must be bound. Bindings can be set by choosing from a set of icons. Control of the advice process is achieved through a VCR-like control panel. Advice is displayed in the form of graphical plan networks with expandable node icons.

6.1 Browsing

The knowledge base is shared by all users of the system. All object icons, activity icons and scenario icons reside there. Two windows display the entire knowledge base. One window, labeled Empty Forms, shows the icons representing classes. In our interface we present it as a reservoir of empty forms which are ready to be filled out. The second window, labeled Filled Out Forms, shows icons for the instances of those classes. Each window is actually a frame surrounding part of its two-dimensional plane. All available icons reside on these planes (figure 29), and may be examined by the users.

6.1.1 Scrolling

Because there are more icons on these planes than can be seen in the windows, users can scroll these planes. This allows the users to see icons that are outside of the frame of the window. Users can scroll the planes horizontally and vertically by employing scroll bars on the edges of the windows, as shown in figure 29.

Another way to browse allows users to "fly" over a plane as if from a great height and see a bird's eye view of the entire plane. The entire plane is represented on a small scale and all available icons appear in miniature. To start

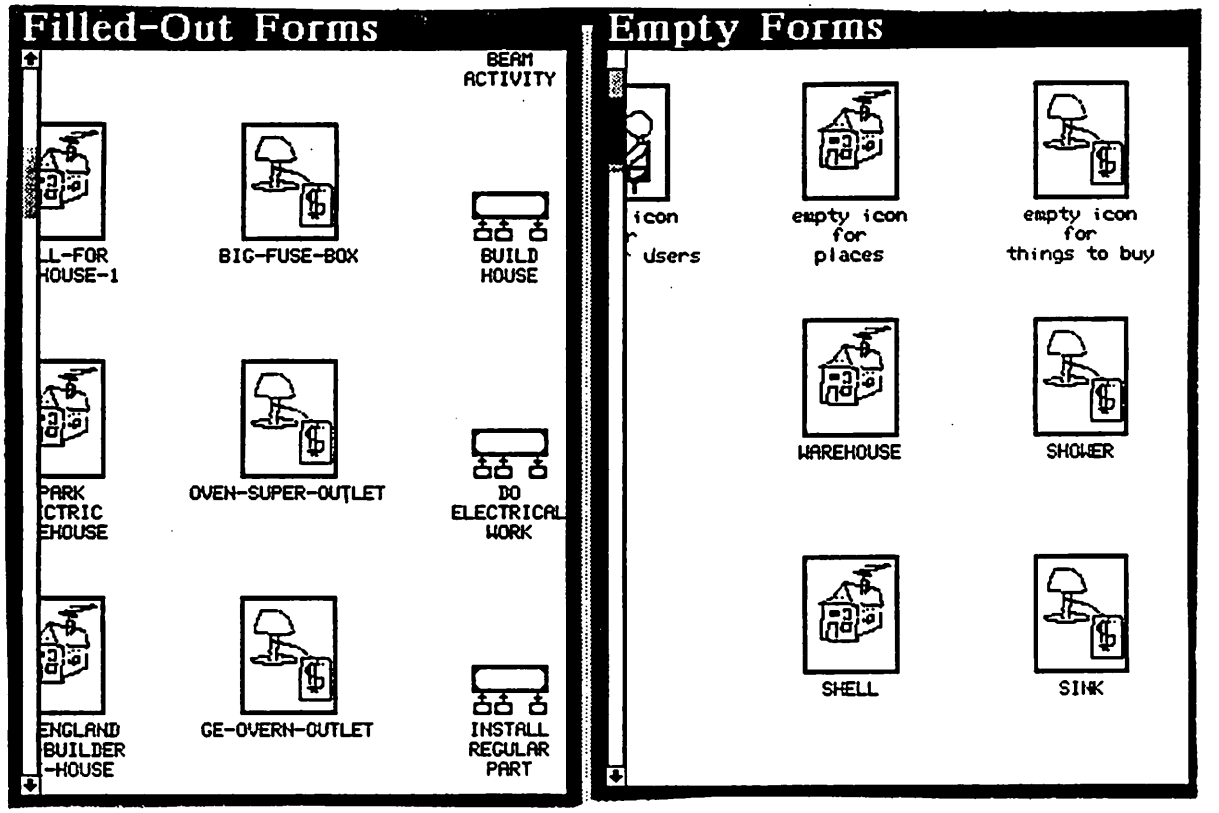


Figure 29: Knowledge Base Windows.

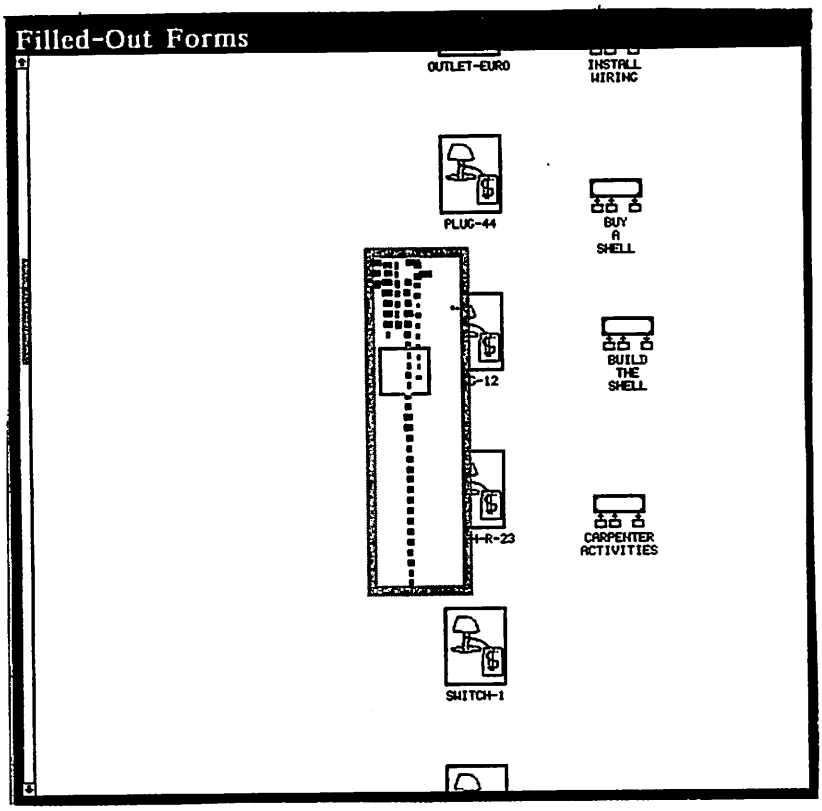


Figure 30: Flying over Knowledge Base.

"flying" users click the middle mouse button, which results in the situation shown in figure 30. The current position of the frame of the window over the plane is indicated by a thin rectangle. Moving the mouse results in moving the thin rectangle over the pattern of icons. Clicking again returns to the regular display mode. The icons within the window frame's current position are displayed at full size. This second way of browsing allows users to quickly retrieve portions of the knowledge base cued by location of groups and patterns of icons.

Icons can be moved in the knowledge base by clicking the middle button of the mouse on the icon to be moved. The icon is then grabbed, and subsequent

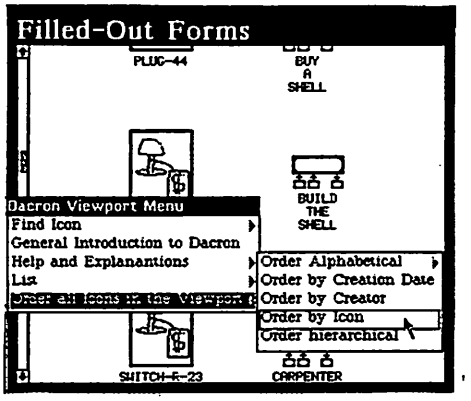


Figure 31: Menu on Knowledge Base Windows.

mouse movement moves the icon on the plane. Clicking the middle button a second time releases the icon at its current position on the plane.

6.1.2 Reordering

The users can reorganize the pattern in which icons are organized on the plane. This function can be invoked by clicking the mouse on the plane that is to be reorganized. A menu appears that allows the users to choose from a number of reordering patterns (Figure 31).

The first option is to order all icons alphabetically by label, starting at the top left corner of the window and continuing left to right, row by row, until all icons are ordered (Figure 32). The second option allows to order by bitmap. All icons that have the same bitmap are ordered in one column. Columns are placed next to each other (Figure 33).

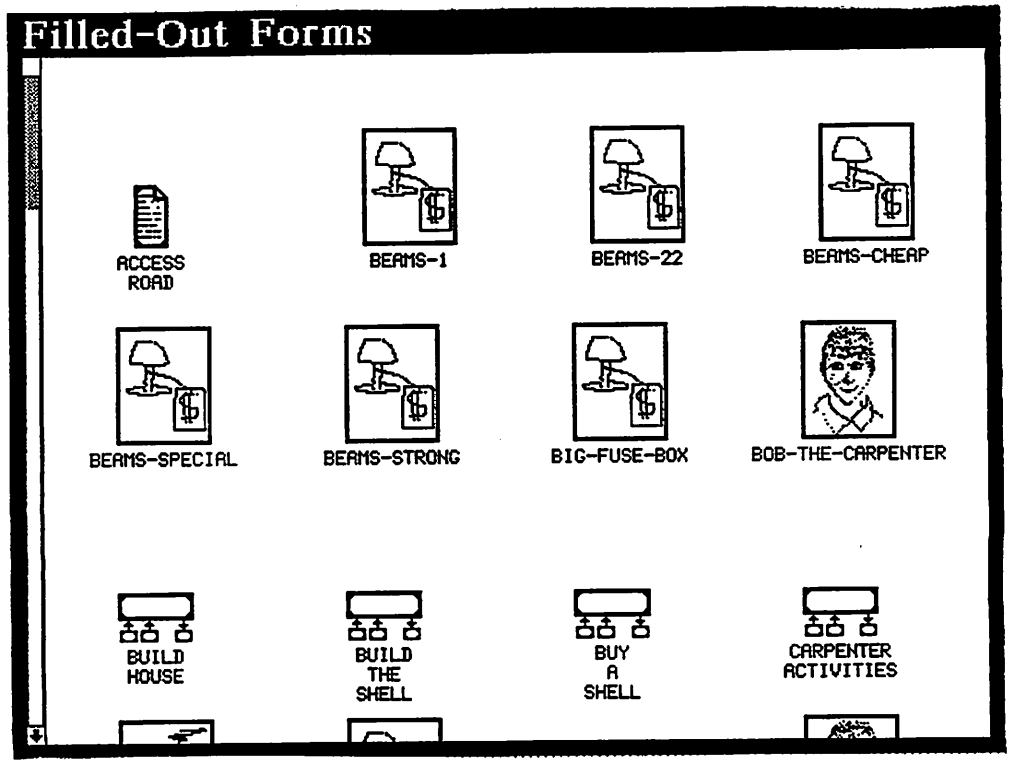


Figure 32: Knowledge Base ordered Alphabetically.

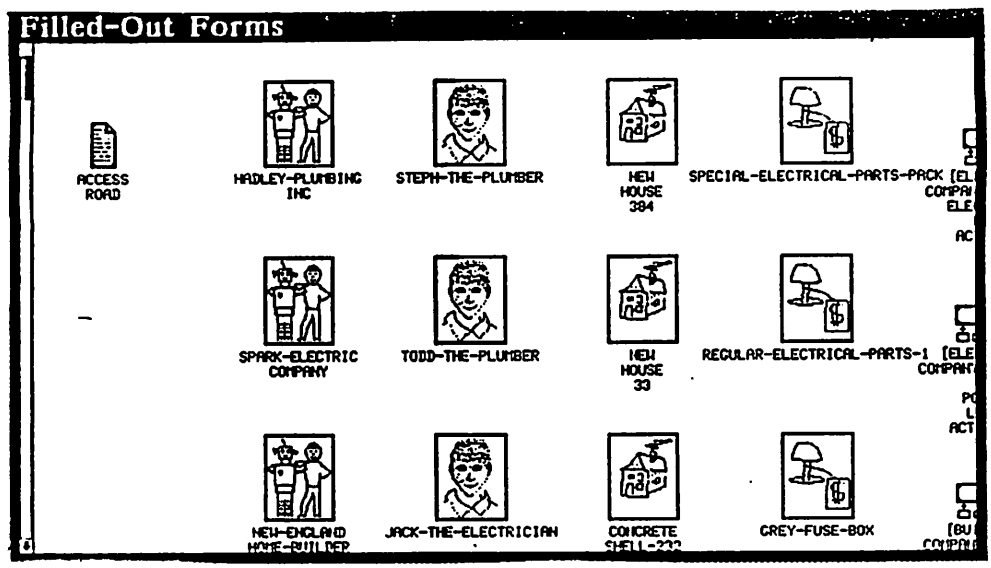


Figure 33: Knowledge Base ordered by Bitmap.

6.2 Searching

Browsing is often too undirected and time consuming for finding a desired icon in the graphical knowledge base. At other times users might not know if a certain icon exists at all. Users might also be interested in finding a set of icons that satisfies certain conditions. Searching functions allow users to access icons quickly and directly.

To locate an item, the users select the `find icon` option from the menu shown in figure 31. A cascading menu allows them to specify how they want to search for the icon. A general way to search for an icon is by its name. There are also options to search for icons by type. Users can also search for icons that satisfy certain properties of activities, objects or scenarios.

Once the query has been specified, either by typing a name, or by specifying an iconic wff, the knowledge base navigation facility will open the finder window to show the result of the search. All icons that satisfy the query are displayed, ordered in columns in the finder window. If no icon can be found, a bell sounds and a message appears in the suggestions window informing the users that no icons could be found.

6.2.1 Searching Alphabetically

Choosing the `Search Icon by Name` option from the search menu results in a prompt to type the name of the icon. Users can now type the name of any unit they want to find in the knowledge base. Upon hitting the return key, DACRON starts to search for icons that carry the typed name as a label.

If such an icon is found, a copy of it is presented in the finder window. If no icon carries the typed name as a label, the suggestions window so informs the users.

6.2.2 Searching for Objects

Interface design guidelines state that semantic knowledge is better retained than syntactic knowledge. We therefore allow the users to search semantically, and not only by name. Selecting the `Search by Components - Places, People, etc.` option of the search menu allows the users to find object icons by their position in the class hierarchy, the slots they possess, the values of slots and the value classes. These choices are made in the last menu of the cascade.

Selecting the `built from` option shows users a prompt that looks like the `Built-From` row in the object editor. Users can copy icons into this row to specify a search for an object that was created from those copied icons. When all desired icons are copied into the prompt users can click on the prompt to start the search.

Users can at all times abort a search specification. They are given a choice to start a new search specification or to exit the search facility. If no icon can be found, a bell rings and the suggestion window notifies the user, like during alphabetical search.

Selecting the `more specialized form` option works in the same way as the `built from` option. In this case users copy icons that are subclasses or members of the desired icon.

Searching by row name results in a prompt that allows users to type in the name of an object's row. Upon hitting the return key the search begins and all objects that possess the specified row are returned as icons in the finder window.

To find icons that have a certain value class users may choose the `value class` option and copy the desired value class icon into the prompt. More than one icon can be copied. When all desired icons are copied to the prompt users can click on the prompt to start the search. DACRON return all icons that have those valueclasses in them.

Icons that contain a specific value in one of their rows can be found by choosing the `value` option. A pop-up menu queries users whether the value is text-based, e.g. "5" or "John", or another icon. An appropriate prompt is then presented to allow users specify the desired values. Upon completion of the specification the search starts in the results are displayed in the finder window.

6.2.3 Searching for Activities

Icons for activities can be found by selecting the `find icon by component - in a plan` option of the search menu. Users can choose among five components that should be addressed by the desired activity. These five components are the goal, the precondition, the decomposition, the effects and the constraints.

Upon selection of the component a prompt appears that looks like the corresponding compartment from an activity editor. A goal compartment appears if activities with a certain goal are to be found, a situation-before compartment appears in the precondition case etc.

These compartments are empty and users can start to specify an iconic wff. Section 6.4.2 explains in the context of knowledge specification, how wffs are specified iconically. This wff is then matched against the wff's in the goal, precondition, effects and decomposition slots of all activities in the knowledge base. If the wff's match, DACRON puts a copy of the activity icon in the finder window.

6.2.4 Searching for Scenarios

Scenarios can be found by specifying their position in the hierarchy of scenarios or by certain facts they contain. These specifications are done in ways similar to those employed in the search for activities and objects.

If the users choose the `built from` or the `more specialized` option from the `scenario` search option, rows for either the more general icons or the more specialized ones will appear. As in the object search case, users can then copy the icons that represent the higher or lower classes in these rows. The search is initiated as for the object case, and the icons that satisfy the conditions are presented in the finder window.

Searching for scenarios that contain one or more certain facts can be done by choosing the by `facts` option of the scenario search menu. A fact compartment appears, that looks similar to the fact compartment in the scenario editor. User specify one or more facts as iconic wff's (see section 6.4.2). When all facts are specified, users can start the search process by clicking in the compartment. The scenarios that contain the desired facts are displayed iconically in the finder window.

6.3 Viewing

All icons in the knowledge base windows, the clipboard and the finder window can be opened to examine the details of the unit. Users can click on the icon and choose the `Look at this Form` option from the icon menu. An appropriate editor appears for that type of unit.

If an editor for that icon is already open, a message appears in the suggestions window to inform the user. The window that contains that editor is brought to the foreground of the screen if it was hidden or overlapped by other windows. In any case, the users are shown the content of the unit.

The window for the editor can be moved around by clicking on its title bar and moving the mouse. Object editors can also be reshaped. Users can scroll up and down in object editors but not in scenario or activity editors, since it would destroy their carefully designed lay-out, as explained in chapter 5.

Once the editor is needed no longer, users can click in the title bar and select the `close editor` option from the menu. The window for the editor is then deleted.

6.3.1 Viewing Objects

In the object editor users can immediately see the rows of the form this editor represents. They can also see the icons for the value classes that are permissible in the rows.

If the users want to see the actual value of a row, they have to click on the value class icon. From this menu they choose the `values - show` option. A

value holder is added to the class icon and the value itself, a string or an icon, is displayed in the holder.

Though the position of the particular object in the class hierarchy is not a slot, it is displayed and manipulated in the same way for reasons of interface consistency.

The value holder can be left open and is closed automatically when the editor is closed or the users can click in the value holder get rid of it. This allows users to concentrate on the parts of an object they think are important.

6.3.2 Viewing Activities

Choosing the `Look at this form` option of an activity icon results in the display of an activity editor. Again there can only be one editor for an icon to avoid versioning problems and ambiguity.

The editor consists of the four compartments discussed in chapter 5. Only the icons representing the subject of a wff are immediately displayed. This allows users to concentrate on the desired parts of the viewed activity.

To see the complete iconic wff users can click on the subject icon and select the `values - show` option from the menu. Though we are not displaying values but wff's, we try to stay consistent with the labeling of menu options used in the other editor types.

DACRON visualizes the wff by creating a value holder for the top predicate of the wff and showing the object icon. All predicates on the subject and the

object are displayed as strings in the subject icon holder and the object icon holder respectively.

Constraints on the icon, be it a subject icon or an object icon, can be seen by choosing the `constraints - show` option of the icon menu. Since constraints are also expressed internally as wff's, the same display mechanism is used for values and constraints.

To close the iconic wff, retaining only the subject icon, which restores the initial state, users can click on the subject icon and choose the `values - close` option. Other ways to close the iconic wff are to click on the object icon holder or the top predicate holder and choose the `close` option.

Icons in the editor can also be moved around. Users click on the icon with the middle mouse button, move the mouse and release the icon by clicking again. Icons can not be moved outside their respective compartments. This prevents confusion over which icon belongs where.

The components of an iconic wff also cannot be moved separately. Trying to grab a single component of an iconic wff and moving it results in the whole iconic wff being grabbed and moved as a unit. This prevents confusion over which holder belongs to which subject icons.

6.3.3 Viewing Scenarios

Opening a scenario icon results in the display of a scenario editor. The functionality for this editor is a mixture of the functionality for the object editor and the activity editor.

The two top rows mean the same thing as the two top rows in the object editor: position of the unit in the hierarchy. They are manipulated as in an object editor.

The compartment for facts contains the subject icons that represent the wff's. These icons can be opened by clicking on them and selecting the `values - open` option from their menu, as in an activity editor. Showing constraints and closing the iconic wff's is achieved in the same way as it is in an activity editor.

6.4 Specifying Units in the Knowledge Base

The specification of knowledge base units, activities, objects and scenarios, is viewed in our metaphor as filling out forms that describe the unit. The Empty Forms window contains all forms that could possibly be filled out to describe a new unit. There are icons representing empty forms for activities, scenarios, the top level object classes and other object classes that exist in a given domain.

New empty forms can be created from existing empty forms. Users can also modify existing classes or instances.

Every full specification process is started by finding the icon for the desired empty form. Clicking on that icon shows its attached menu (figure 34). To create an instance of that class, users choose the `Fill out this Form` option. An empty editor for that type of icon appears.

A prompt appears in the editor asking the users to type the name of the unit. Upon completion of the name, the prompt disappears and the new name appears in the title bar of the editor.

DACRON creates a temporary unit at this time, corresponding to the "form" filled out by the user. The corresponding unit is specified in terms of the underlying knowledge representation language, POLYMER in this case. This unit is temporary because it is not in the knowledge base and can be easily deleted.

Once the empty editor with the new name is created, users can fill out the "form". They can copy icons to the rows or compartments of the editor or choose to type string values. The particular interactions applicable in each editor are introduced below. Every specification of a value in the editor is not only represented graphically as a result of the manipulation by users, but also translated into the underlying POLYMER formalism, and recorded in a temporary DACRON structure of the unit.

Values in the rows and compartment can also be completely deleted or modified during the specification process. The temporary unit is updated accordingly. Deletion or specification of values is not possible when viewing the unit. This prevents users from making unintentional changes. If such an operation is ever attempted when only viewing a unit, a message informing the users will appear in the suggestions window.

When users have completed a new form, they click on the title bar of its editor. From the editor menu users can choose the Save Form and Close Editor option. This option closes the editor's window, and submits the new unit permanently to the knowledge base, where it is represented by a new, highlighted

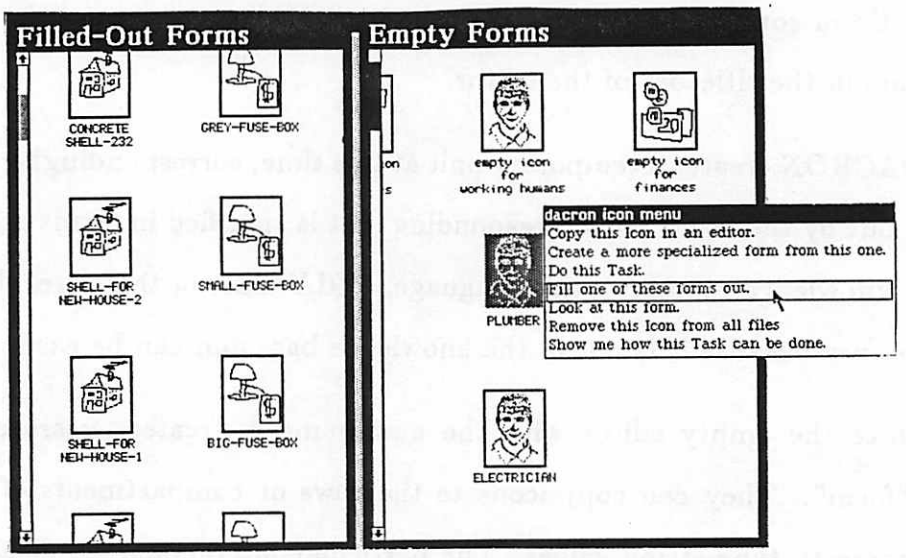


Figure 34: Menu on Icon for Empty Form.

icon which appears in the middle of the Filled Out Forms window. Alternatively, users can choose the Abort - Undo Everything option, which will also close the editor window. This operation will not save the temporary unit but deletes it to restore the status ante quo.

6.4.1 Iconic Object Specification

Object specification starts by finding the icon for the empty form that corresponds to the object users want to describe. Objects are different from activities and scenarios in that many icons and classes for objects exist, e.g. *Agent, Human.Agent, Written.Material*, while there is only one icon type for scenarios and only one icon type for empty forms describing activities. This object particularity makes it necessary to allow the users to create new empty forms (classes).

6.4.1.1 Creating New Classes

A new class (new empty form, new icon in Empty Forms) is created by finding the icon that represents the closest existing form. To create a form for craftsmen for example, users could start from the Human.Agents icon. They would choose the `Create a more specialized Form` from the icon menu (figure 34).

The new editor appears and the new title is assigned as already described. All rows present in the initial form are present; they are inherited through the hierarchical object structure [TL82]. New slots for the new object class may be specified. To do so, users add new rows to the existing form.

A pop-up menu queries the users whether or not a row should be added. Upon confirmation, the editor window is scrolled toward the bottom and a new, unnamed row appears after the last inherited row. A cursor is positioned in the top left corner of the row and a prompt in the row asks the users to type the name of the new row. Having typed the name, the prompt disappears, the new row name appears in its place, and the users are asked whether another row should be added. This process is repeated until no more rows are to be added.

The creation of every row in the graphical editor results in the creation of a new slot in the underlying, temporary unit. No specific value class is initially associated with the slot. Users can set the value class by copying one of the value class-icons from the value class window to the new row.

The new class is submitted to the knowledge base as it was for the general case, by selecting the "save" option form the editor menu. A new icon representing the new form appears highlighted, in the middle of the Empty Forms

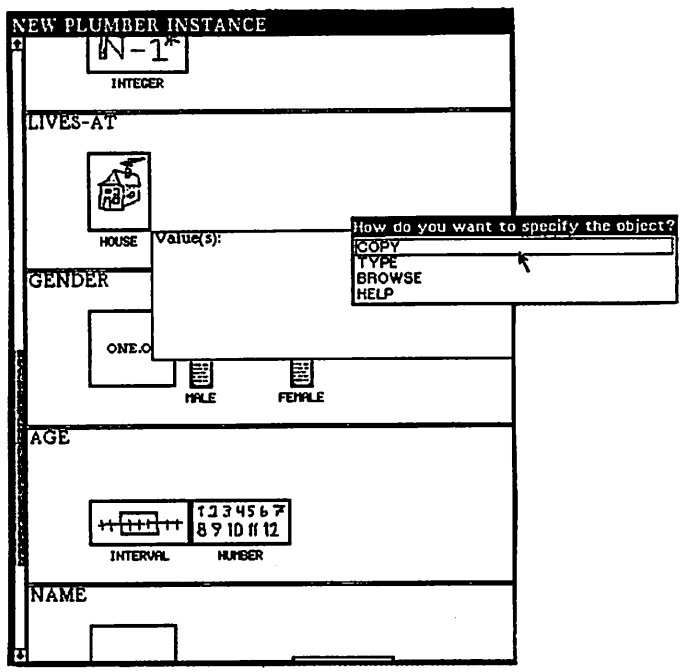


Figure 35: Pop-up Menu: Type or Copy to Set Value.

window. The bitmap for the new icon is the same as the bitmap for the parent class. The label is the taken from the title of the editor.

6.4.1.2 Creating Instances of Classes

Filling out empty forms for objects results in the creation of instances of that class. Opening the editor and naming an instance are done in the general way.

To specify the value of a certain row, users click on the value class icon in that row and select the `values - specify` option. The users are then asked whether they want to type a numeric value or a string via the keyboard or if they want to copy another icon (figure 35). After that decision a value holder appears next to the value class icon, ready to accept input.

In the case of copying another icon, users can simply go to any icon in the knowledge base and copy it to the value holder. If users want to type, a cursor appears in the value holder and users are prompted to type. The value holder stays open, showing the new value, until explicitly closed by the users. The holder can then be reopened to display the new value.

The specification of values is immediately translated to the underlying unit. The particular slot concerned is determined, the value is translated into formal language terms, and attached to the slot.

If the value is not in the limits for the value class on the slot, it is rejected. The value disappears from the value holder and the suggestions window displays a message explaining the problem. Users can then try to specify a legal value.

6.4.2 Iconic Activity Specification

Specifying activities means filling out an empty form for an activity. Again users start by selecting the `Fill this Form out` option of the icon for the empty activity form in the Empty Forms window. The values that go in the compartments of the form are iconically specified wff's. Users can start with any compartment they like. There is no binding way to fill out the form.

6.4.2.1 Iconic Specification of Wff's

The value of every compartment, be it a goal compartment, a compartment for the preconditions, the actions or the effects, has to be a wff. This wff is specified iconically. Users start by copying the icon for the subject of the wff to the compartment.

Take, for example, the house construction domain. In order to specify that the *New England Home Builder Inc.* is in possession of *Housing Project 88* users starts by copying the icon for the builder company to the goal compartment. Figure 36 shows this situation.

To select the top predicate of the wff, which slightly resembles a verb in a sentence, users click on the subject icon and select `values - specify` form the menu. The option was named this way to achieve congruency among menus in different types of editors.

Selecting this option results in the display of all possible predicates that could be specified for the subject icon. This menu constrains users' choices and therefore helps them to decide, while keeping the knowledge base coherent.

If none of the displayed predicates seems right to users, they can choose the `specify another one` option. This option allows the users to update the entire class the subject icon was taken from without interrupting the iconic specification process. Choosing this option displays a prompt where users can type the desired predicate. The underlying definition for the class of objects concerned is augmented with the new predicate as a slot.

Figure 37 shows the menu of predicates for the New England Home Builder Inc. `In.Possession.Of` is one of the options. Users can now choose this option with the mouse.

The chosen predicate, either taken from the menu, or typed in, appears as label in a predicate value holder that is created next to the subject icon. An object predicate holder appears also. The object predicate holder is empty. Users are asked if they want to type the object value or copy an icon for it.

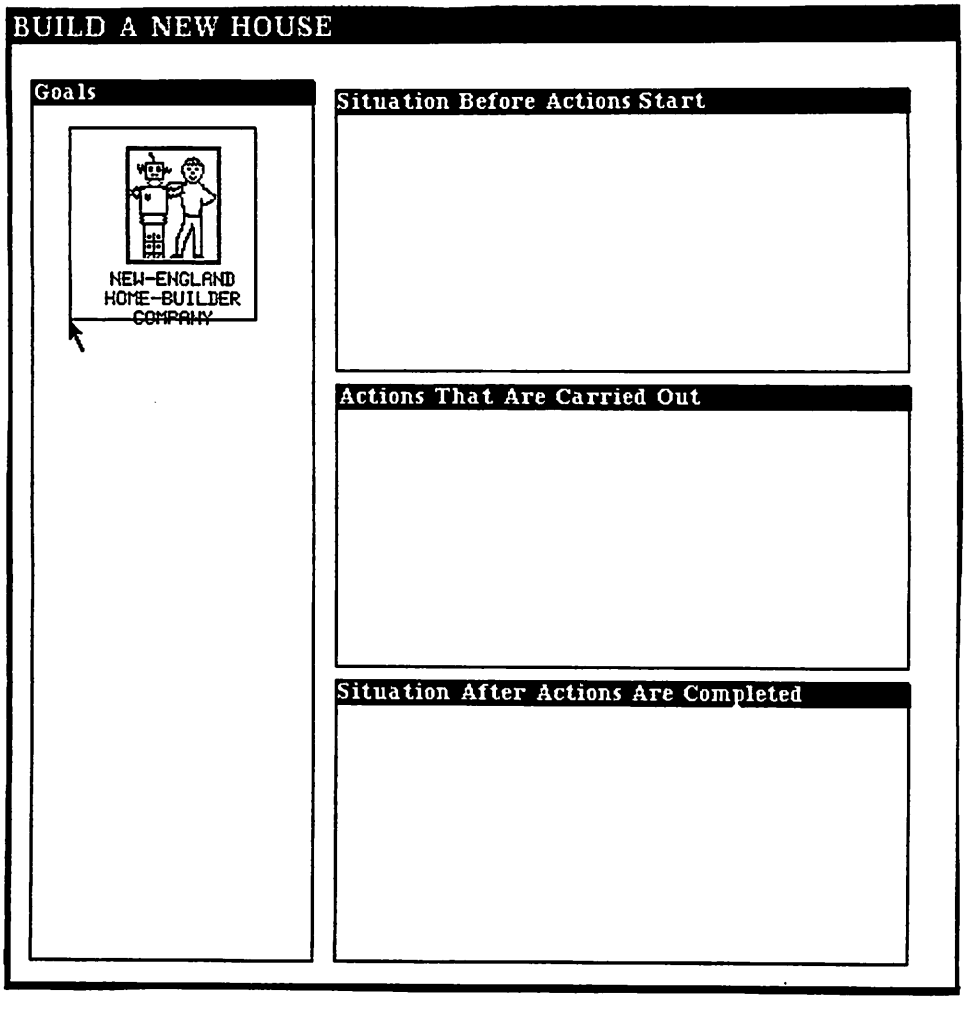


Figure 36: New Editor with Subject Icon.

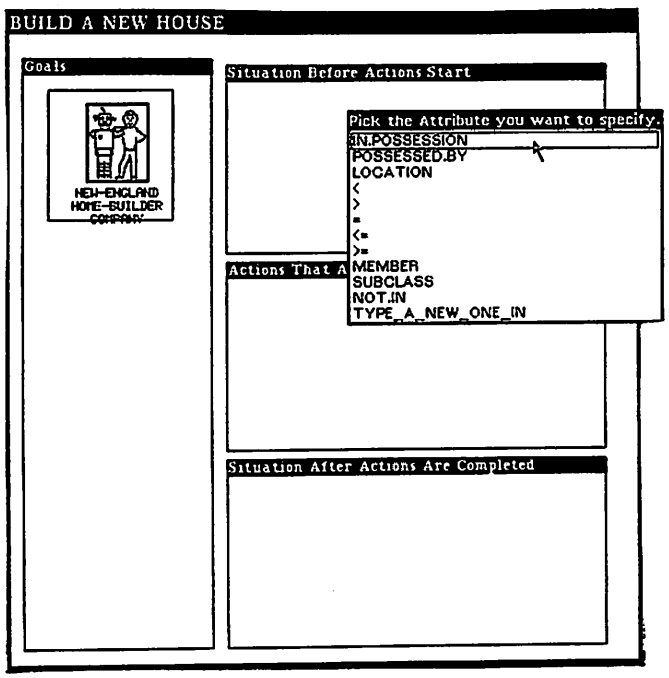


Figure 37: Choosing the Top Predicate.

If users choose to type, as for the wff ''`price house $120.000`'', a prompt appears in the object predicate holder and users can type the desired string or number. If users choose to copy they can select any icon and copy it to the object holder.

A third option allows users to browse and search in the knowledge base before deciding whether to type or to copy. This gives users the chance to find an appropriate icon before making a commitment.

The help option explains what can and should be done to specify the object icon. The whole process of iconic specification is complemented by messages in the suggestions window that display textual advice about what to do next in the prototypical case.

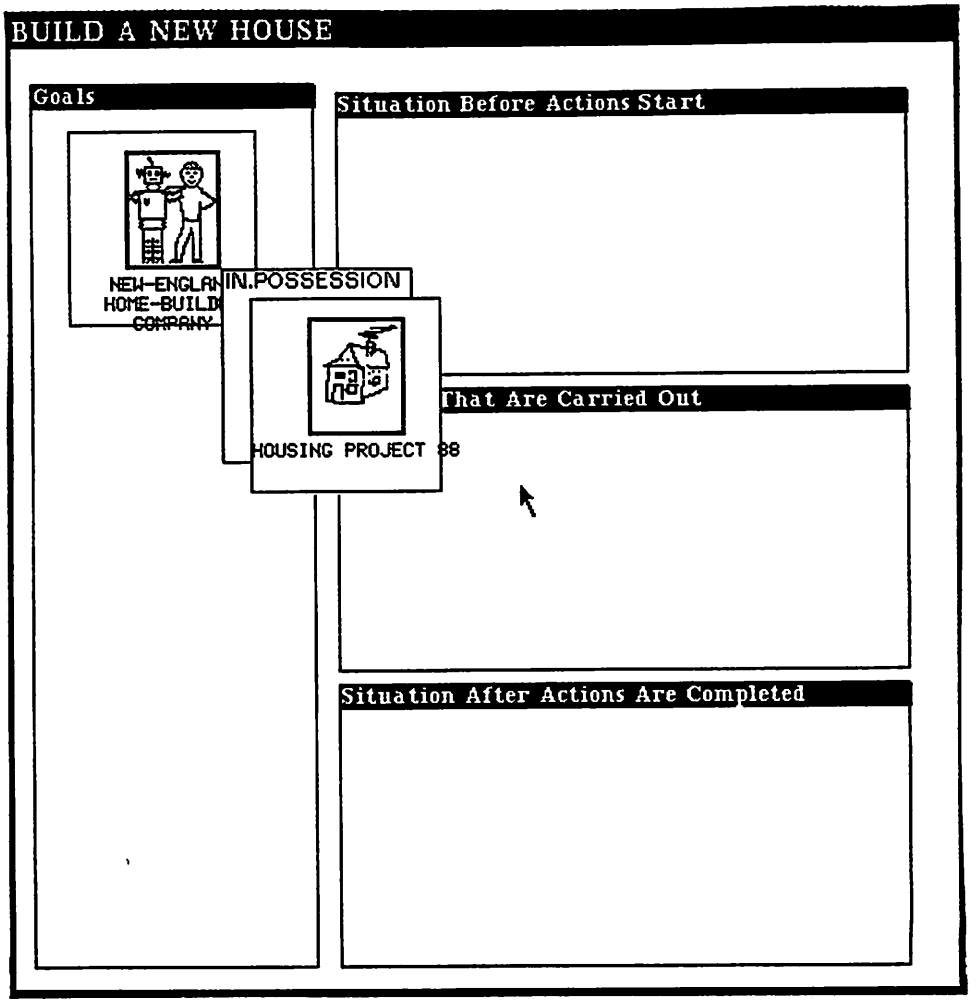


Figure 38: Completely specified Wff.

In our example, users would choose to copy and bring the icon for `Housing.Project.88` into the object holder. Figure 38 shows the new situation.

To specify additional predicates for the object or the subject, users can open the menus for those holders by clicking on them with the mouse. They then choose the option **specify predicates**. As in the case for the top predicate, a menu of predicate options appears. Users now choose to specify complicated wff's like `(owned.by (land housing.project) (father (mayor Deerfield)))` expressing that the land for the housing project is owned by the

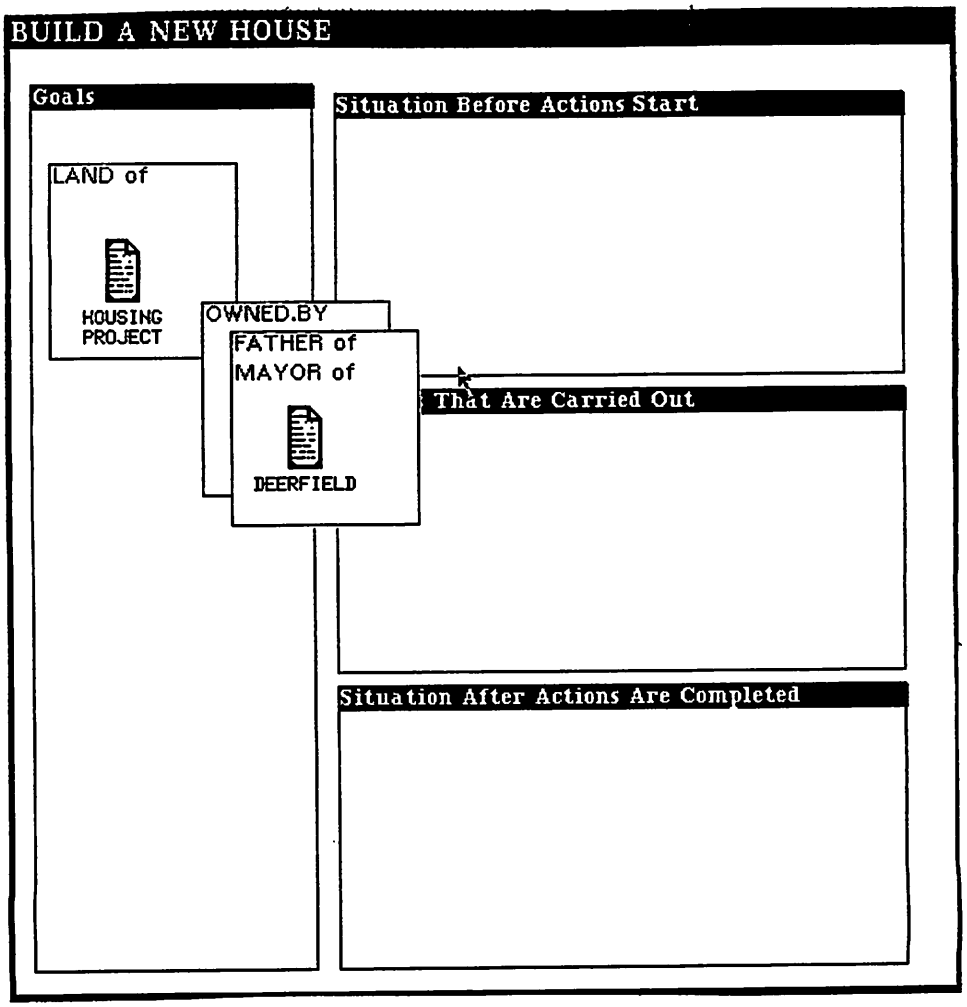


Figure 39: A complex iconic Wff.

father of the mayor of Deerfield. The predicates that pertain to an icon are listed in its holder (figure 39).

Users can not only copy icons but whole wff's. Preconditions in an activity might be very similar. The precondition `(have ?plumber ?pipes)`, `(have ?plumber ?joints)` and `(have ?plumber ?wax)` for example consists of three wff's that only differ in the wff-object. It would be annoying for the users to go through the completely same specification process for all three wff's. Users can instead copy the complete first wff. They can then delete the object icon

and insert the desired one. Wff' can be copied between compartments and also between editors.

To copy a complete wff users choose the `copy - whole sentence` option form the subject icon menu. A duplicate of the wff appears that can be moved to the desired location by moving the mouse and clicking at the end of the move.

6.4.2.2 Variables in Iconic Wff's

It is necessary to allow users to specify wff's that contain variables. An examples is `(installed ?wire ?house)`, to express that a wire, to be bound to a particular wire at run time, is installed in a house. Wire and house are used as names for variables, indicated by the question mark.

To specify variables users can not use the icons that represent instances. Instances refer to a particular object. This leaves us with the icons for classes in the Empty Forms viewport.

Inserting an icon for an empty form in the iconic specification means that this type of form should be used when executing the plan. It automatically constrains the variable to a certain type.

Using icons for empty forms that are to be replaced by filled out forms at execution time also fits the form metaphor. Users can imagine this as the machine filling the empty form when the time comes.

To specify the wff `(installed ?a.wire ?a.house)`, where `(member ?a.-wire (class WIRES))` and `(member ?a.house (class HOUSES))`, users would copy the icon describing the empty form for the WIRES class in the Empty Forms

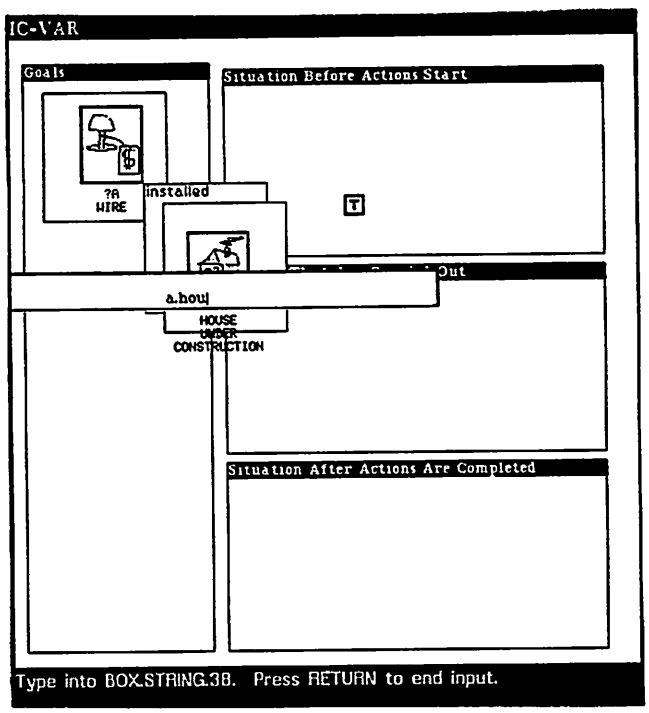


Figure 40: Labeling a Variable.

window to the desired compartment in the editor. Upon copying, a prompt appears in the label of the icon that allows users to type the name of the variable, `a.wire` in this case (figure 40). DACRON adds the question mark automatically to show that this icon represents a variable and not an instance. DACRON also creates the member-constraints automatically.

Users can then proceed as for the general wff specification, choosing a predicate from the icon menu and copying the icon for the object into the opened holder.

6.4.2.3 Specification of Decomposition

The steps in an activity can be specified in two ways. Users can either copy existing activity icons from the Filled Out Forms Window into the action compartment or they can copy the icon for empty activity forms if no icon for the step exits.

In the first case users find an activity icon that has a goal corresponding to the subgoal. The users take this icon and copy it to the action compartment of the activity under specification. DACRON uses the label of the copied icon as label for the subtask. The goal-wff of the copied icon's form is taken as wff for the subgoal.

To specify, for example, that a step in building a house is to have a frame, users could copy the Build.Shell icon. Figure 41 shows the Build.Shell icon in the knowledge base window, the open editor for it and a copy of the icon in the actions compartment of the editor for the activity that is currently specified.

If users can not find an icon that fits the subgoal, they may copy the icon for the empty activity form. Copying this icon results in the display of a prompt, consistent with the iconic definition of variables. Users type the name of the subgoal in the prompt. DACRON recognizes that only a labeled icon exists but no wff describing the subgoal. A subgoal holder is opened and the suggestions window informs the users to specify the subgoal.

Users specify the subgoal in the general iconic wff specification way. Once the subgoal-wff is complete DACRON generates a new activity unit. This unit has the subgoal as its goal and the subgoal label as its name. DACRON thereby guarantees that at least one activity description exists that can satisfy the sub-

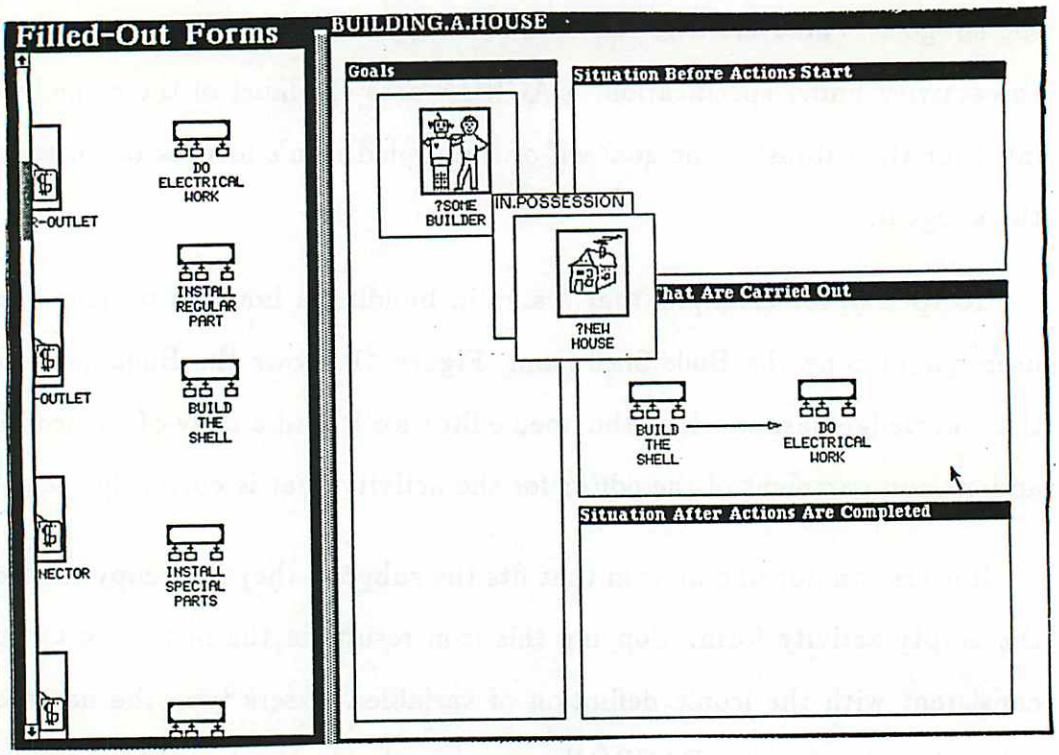


Figure 41: Copying existing Icon for Subgoal.

goal. Figure 42 shows the subgoal wff in the editor for the house building activity and the automatically generated editor for the Do.Electrical.Work activity, generated from the subgoal.

Users can now continue to specify the original activity or they can turn to the newly created one. At this point it would be possible to allow users to delegate the specification of the lower level activity to other users, who have more expertise with that activity. Cooperative knowledge specification, hierarchical and organizational distribution of knowledge could be accounted for in a simple way.

DACRON has no facilities for the distributed cooperative specification of knowledge. To expand DACRON in this direction it would be necessary to have a model of human cooperation and the communication protocols of distributed knowledge-based systems.

If the new unit is not attended to by any user, DACRON turns it into an agent executable action. Agent executable actions are primitive actions that have to be directly executed by human agents and can not be decomposed any further by the system. A message to the agent is generated at run time.

6.4.2.4 Graphical Specification of Plan Rationale

The plan rationale is specified by positioning arrows between icons representing subgoals. An arrow from icon A to icon B be indicates that A has to be achieved before B.

Users click on the action compartment and choose the `Set Arrows` option. The cursor turns into a crosshair. Users click the crosshair on icon A. The

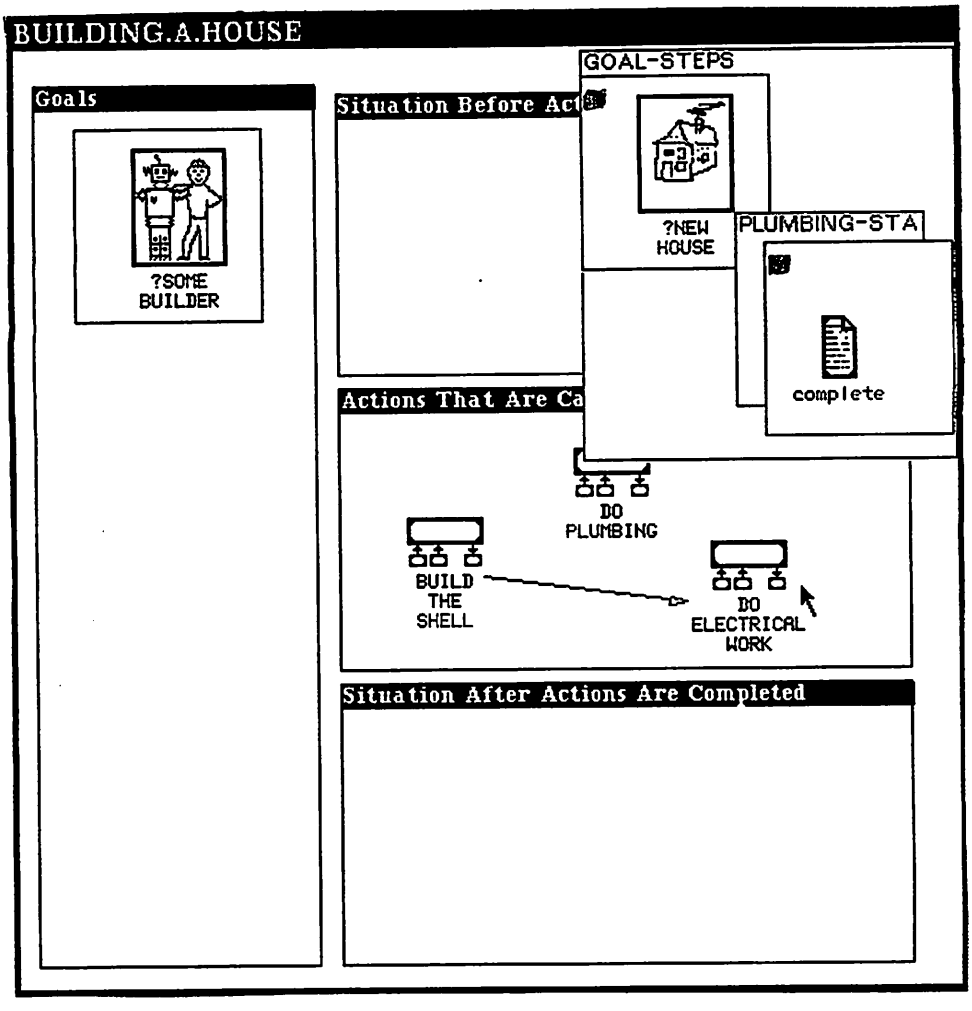


Figure 42: Specifying Subgoal Wff.

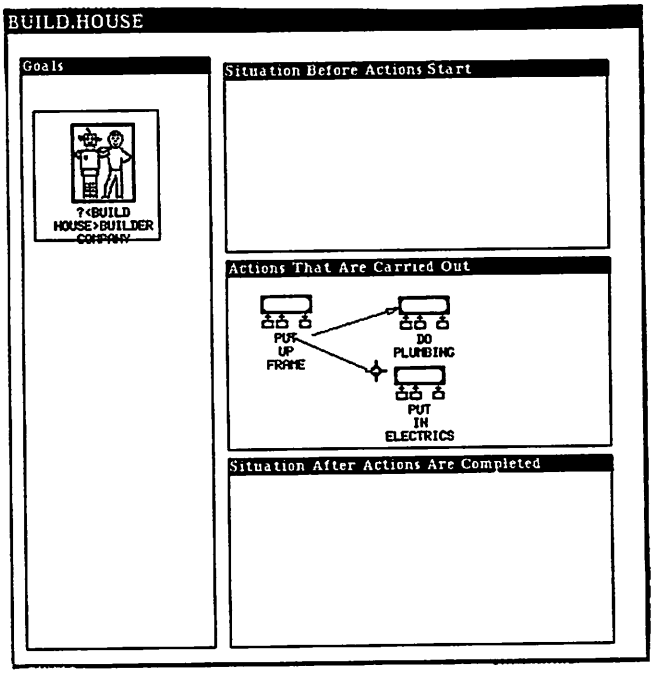


Figure 43: Plan Rationale Specification.

mouse then allows the users to "pull" a rubber band, originating at icon A to the desired icon, B in our example. Clicking again releases the end of the rubber band. The rubber band is then turned into a directed arrow.

Figure 43 shows an example from the house construction domain. Electrical work can only be done when a frame exists. The figure shows the point at which users are moving the rubber band to the second icon.

6.4.2.5 Specification of Effects

The description of effects is a little bit more complex than those for goals, preconditions, etc. In addition to the wff specifying a certain state of the knowledge base, it must be indicated whether the wff should be added to the current scenario, deleted from it, or if an existing wff should be set to a certain value.

The specification of a wff in the situation-after compartment contains therefore an additional step. After copying the subject icon, users are asked if the want to `ADD, SET, DELETE` the following wff.

Picking one of those options results in the creation of a holder that carries ADD, SET or DELETE in the top left corner. The subject icon is positioned on that holder.

6.4.2.6 *Consistency Checking and Additional Elicitation*

Users can, at any time, decide that the specification process is complete and that the new activity should be added to the knowledge base. It is impossible, however, to simply close the editor as in the display case. DACRON will inform the users that they must either save the units to the knowledge base or explicitly give the UNDO/ABORT command to remove the editor, its contents and attached units. To give one of these commands, users click in the title bar of the editor and choose from the menu.

If a unit is to be added to the knowledge base, the consistency checker is invoked. The functionality of the consistency checker is described in chapter 5. If no inconsistencies are detected the new activity will be added permanently to the knowledge base. The suggestion window displays a message informing the users of that fact, the editor is closed and the new icon appears, highlighted in the middle of the Filled Out Forms window.

In the case of inconsistencies (circular plan rationale, wff contradiction, variable introduced in effects, empty compartments, effect handling) the concerned

wff or compartment is highlighted and a message appears in the suggestions window. Users can then debug in the editor.

If other inconsistencies occur, they are reported in the same way. The process is repeated until all inconsistencies are removed. The users can then add the unit to the knowledge base or decide to abort specification.

Then a pop-up menu appears asking the users if there are alternative ways to achieve the goals in the activity added. An unnamed editor is opened that shows the previously specified activity with all wff's removed but the goal. These empty compartments suggest the absence of familiar preconditions and decomposition to the users. The goal wff shows them the same goal, inducing recall.

If the users answer NO to the pop-up menu, the editor disappears. In the confirmative case a new activity specification process can begin in the open editor.

6.4.2.7 Creation by Modification and Updating

It is not always necessary to build activities from empty forms. Existing activities may be very similar to new activities users want to specify. Just as with iconic wff's, users can built a new activity by starting with an existing one.

Users are also allowed to make changes to existing forms describing activities. Updating an activity in this way is similar to creating a new activity by modifying an existing one under a new name.

In either case users start by opening the menu of the existing activity icon. They then choose the `Specify` option from it. The editor for the activity is

opened and a pop-up menu lets the users choose either to create a new activity or update the existing one.

If users decide to create a new activity, a prompt appears that asks them to type a name for the new activity. This name replaces the old label in the title bar of the editor. Users can then specify additional iconic wff's. When updating an existing activity users can delete and specify icons and wff's to express the changes of the activity.

When the updating or modification process is completed, users save the activity by selecting the `save unit and close editor` option, as for the regular specification case. The consistency checker is invoked. The process is same as in the regular specification case.

6.4.3 Iconic Scenario Specification

Like object and activity specification, scenario specification starts by finding the icon for empty scenario forms in the Empty Forms window. The actual specification process follows the form filling metaphor.

The scenario editor, consisting of parts of the object editor and parts of the activity editor, can be treated like an activity editor. The two rows indicating the class hierarchies that this editor shares with the object editor are not manipulated directly. Their values derive from the unit's position in the hierarchy. This leaves the compartment for facts which can be filled with iconic wff's for the set of facts comprising the scenario.

An example from the house construction domain is the scenario of building a new house in a rural area. The facts defining this scenario are:

```
(level ?ground.water high)
(connected ?lot electricity)
(connected ?lot ?road)
```

Appropriate constraints for the variables are:

```
(member ?ground.water (class GROUND.WATER))
(member ?lot (class LOTS))
(member ?road (class ACCESS.ROADS))
```

To specify the rural construction scenario users choose the `fill this form out` option from the icon for empty scenario form. The empty editor for scenarios appears. Users are asked to type the name for the new scenario into the prompt.

The users can then copy the icon for the class of lots. Upon copying, the name of the variable can be assigned at the prompt. Then users open the menu of the icon and choose the `connected` predicate. A holder appears with the label *connected* in the top left corner of the holder. Users can then copy the icon for electricity. The specification of a fact, expressed by a wff is now complete.

DACRON infers the constraints for the variables automatically from the icon that was used to specify the variable. In the above example DACRON adds `(member ?lot (class LOTS))` to the description of the scenario.

Wff's can be copied from open editors to the fact compartment of a scenario editor. These wff's can be taken from other scenario editors or from other open activity editors. Likewise wff's from scenario editors can be used in the specification of activities.

6.4.3.1 *Modifying and Updating Existing Scenarios*

Icons for forms describing particular scenarios, like the rural house building scenario for example, reside in the Filled Out Forms window. These scenario descriptions can be updated and modified as was the case with activities.

To do so, users choose the `create more specialized form` option of the scenario icon's menu. The editor for the scenario at hand opens and users are asked whether a new scenario should be specified from the existing one or if this scenario should keep its name and be updated.

In the modification case users can give a new name via the prompt and then start adding facts as iconic wff's. In the update case users can start adding and deleting wff's right away.

6.4.3.2 *Merging Scenarios*

A quick way to generate complex scenarios is by combining the descriptions of individual scenarios in a new scenario. The "Rural Construction Scenario" and the "Credit Union Loan Scenario" might be combined to express a scenario in which a house is to be built in a rural area with the financial help of a credit union.

To merge existing scenarios to generate a compound scenario, users have to start with an empty form for a scenario. Instead of copying wff by wff or even specifying the single wff's, users can copy the existing scenario icons in the facts compartment of the new, compound scenario.

The new scenario takes all facts from the scenario icon copied. DACRON transfers the constraints on variables also. Opening one of the scenario icons in the facts compartment results in the display of its constituent facts in iconic form.

Figure 44 shows the editor for the Rural Construction with Help of Credit Union scenario. The for icon the rural construction scenario is in the facts compartment. The icon for the credit union scenario is also in the facts compartment but it has been opened. The facts contained in this constituent sub-scenario can be seen in the figure.

6.4.4 Deleting Icons and Wff's

Activities, objects and scenarios in the knowledge base can only be deleted by removing the corresponding icon from the knowledge base window. This is done by clicking on the icon and selecting the `Remove this icon` option from the menu. Users are asked to confirm the deletion of the unit. The editor for the activity, object or scenario to be deleted, is also shown. Users can either confirm the deletion in which case the editor and the icon disappear or choose to not delete, in which case the editor closes, and the icon in the knowledge base window is highlighted.

This form of deletion must be distinguished from the deletion of an icon in an editor. In the editor the icon might represent the value of a row or the subject or object of a wff. Removing such an icon only means that the corresponding value of the row is deleted or that the part of the wff is removed. It does not mean that the underlying unit is to be deleted.

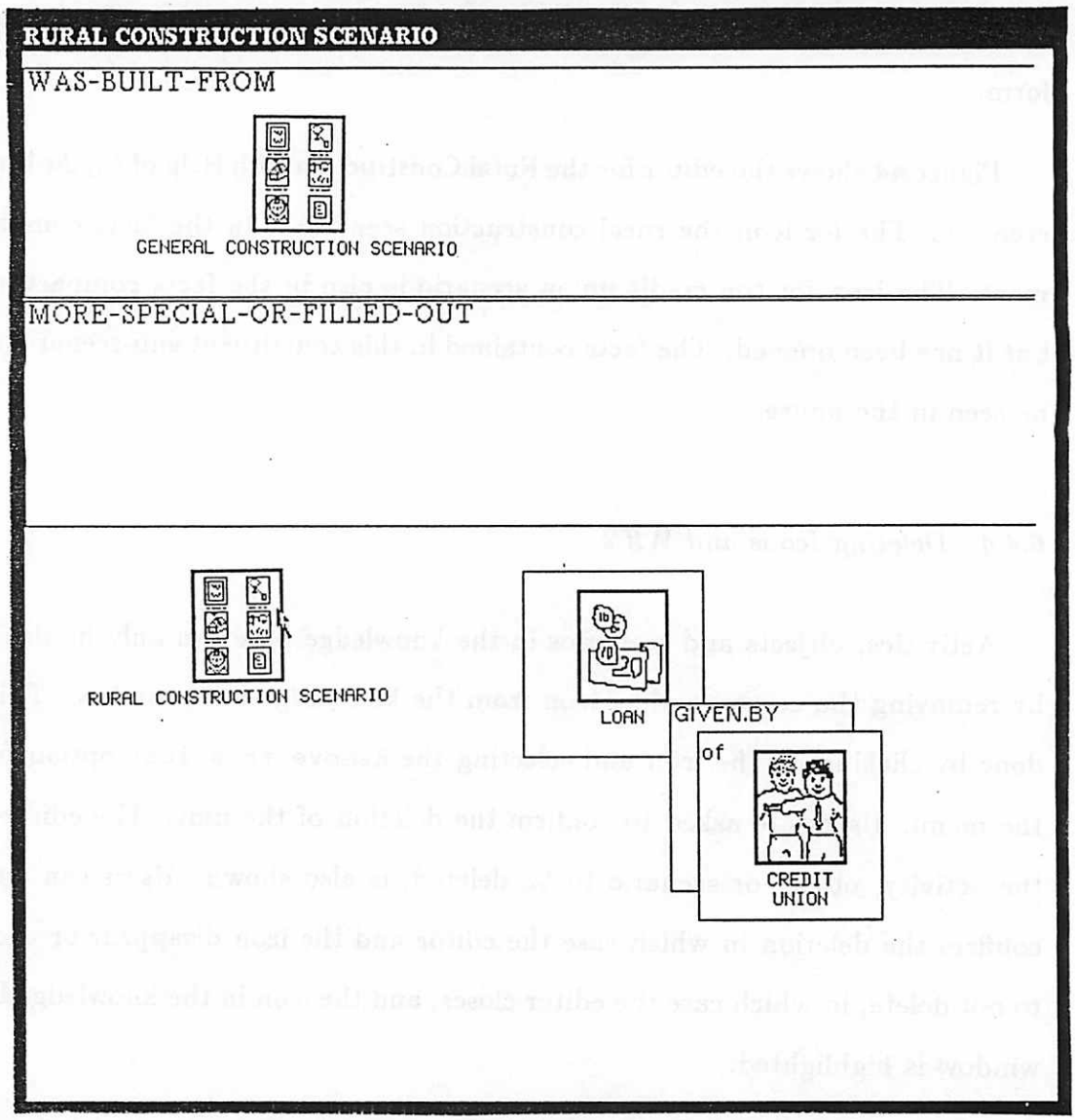


Figure 44: Merged Scenarios.

Deleting House.454 in the Filled Out Forms window completely removes it from the knowledge base. House.454 does not exist any longer. Deleting House.454 in the editor for Jim.the.House.Owner it only means that Jim no longer owns House.454.

Users can delete wff's by deleting their constituent icons and holders piece by piece, or by removing the whole wff in once part. To delete a whole wff users can choose the **delete - whole sentence** option from the menu on the subject icon. If the wff is closed and only the subject icon visible, then that icon is deleted. If the whole wff is open and all holders and the object icon are visible, then the whole picture hierarchy is deleted. The wff is removed as a value from the underlying unit.

6.5 Soliciting Advice

The knowledge that was entered by domain experts can be utilized by all other DACRON users. On a simple level, other users might just look at the knowledge base or at editors for particular forms. But DACRON can also give sophisticated advice on how to do an activity. In that case, DACRON calls the planner and presents graphical version of the generated plan networks as advice to the user.

Because an initialized activity, used to generate or expand a plan network, consists of many different subgoals, decompositions, constraint bindings, etc., it is impossible to display this large amount of information in one single graphical

structure or even on one single screen. We solve this display problem by using time as an additional display dimension.

In particular we have implemented a top-down animation of the planning process. This animation consists of a sequential display of activity editors and graphical plan networks. As each activity editor is on the screen certain variable bindings, values of icons, etc., are displayed. This is done by highlighting an icon in the plan network, and displaying its associated editor. Icons representing variables in that editor are highlighting. Users choose values for that variable from a set of appropriate icons, displayed in a separate window. When all variables are bound users can close the editor. Then the next editor appears on the screen and so on until the whole planning process has been visualized for the user.

6.5.1 Initiating an Advice Session

To get detailed advice about the execution of a certain activity, users find the icon corresponding to that activity. The Filled Out Forms window graphically represents all activities in the knowledge base.

Upon locating the desired icon for the activity users want advice on, users select Show me how this task can be done from the icon menu. For example, in figure 45 a user selected `Build House` to elicit advice on building a house.

To run an advice session, users must also bind variables that occur in an activity and they must select a scenario that supplies that context for the activity. The scenario is the state of the world when the activity starts.

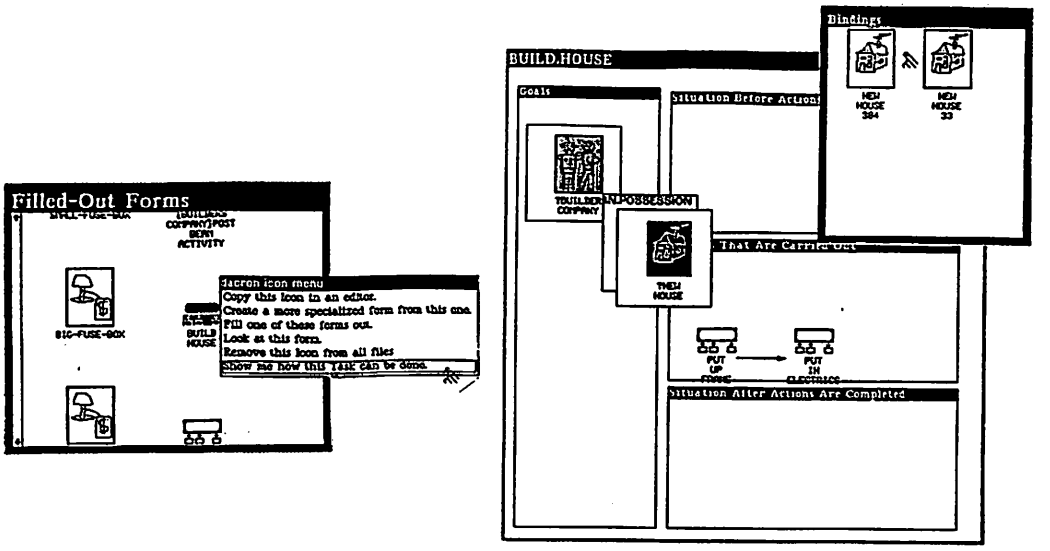


Figure 45: Selecting Task for Advice and Binding Variables

6.5.1.1 Iconically Binding Variables

DACRON asks the users to provide bindings for the variables in the activity description. The activity editor is displayed, and all variable icons are highlighted. When the users click on a highlighted icon, a window appears, displaying the set of icons representing the possible values for this variable (figure 45). Users then choose one of these value-icons with the mouse, and the variable is bound. Thus, the `?New.House` variable is bound to `New.House.33`.

Only icons that satisfy the constraints on the variable are presented as choices. All other value icons are not considered. This guarantees valid bindings, while freeing the users from performing tedious constraint checking manually.

6.5.1.2 *Iconically Selecting a Scenario*

The next step is to choose a scenario in which this activity will happen. A set of icons representing predefined situations appears, and the users select one of these scenario icons with the mouse. Only scenarios that are relevant to a certain activity are presented as choice. Similar to the variable binding method, where only those icons were presented that satisfied the constraints on the variable, only icons that satisfy the preconditions of the activity are presented.

If no such scenarios exist, a message appears in the suggestions window. The users can either specify a scenario that makes this activity executable or they can abort the advice session. Figure 46 shows the editor for the Build.House activity. The window to select the scenarios is open. Users have the choice of picking one of the two scenarios as an initial state of the world. The course of the advice depends on the scenario picked. Building a house in a rural scenario might require the construction of an access road, while the road is already completed in a suburban construction scenario.

6.5.2 *Animating the Planning Process*

After bindings are completed, DACRON displays the initial plan network in a new window. A separate control window shows Controls that represent the users' control over the planning process. These controls are modeled after the familiar image of a VCR control board, with the following functionality:

- The Forward button controls the stepwise refinement and expansion of a plan;
- the Backward button lets the users view the previous version of the plan network;

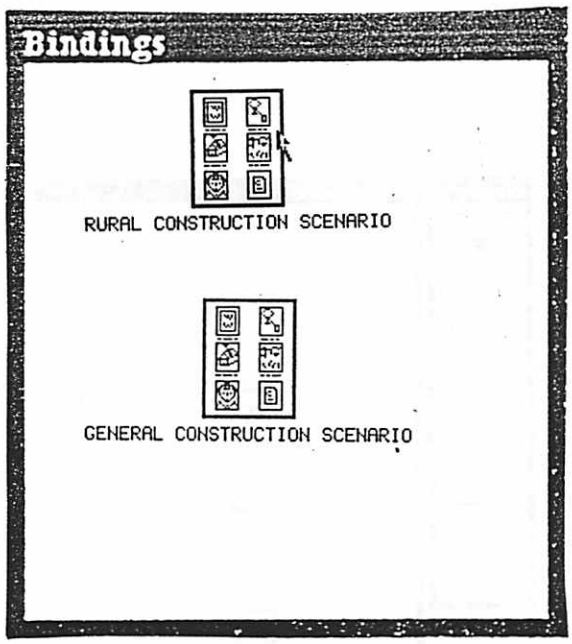


Figure 46: Choice of Situations for Activity.

- the Abort button ends the advice session, and
- the Selections button provides menu-based options of display and information control to the user.

The stepwise refinement of a plan network is the basis of the planning process representation. Selecting the *Forward* button causes a node in the network to flash.

This indicates that the flashing node is currently selected by the planner. The planner tries to find an activity that matches this node. If it finds one, it replaces the node with the content of the activity. In the case of a goal node, that node is replaced by nodes that are the steps of the activity. If the activity represents a tool or agent action, then the node is turned into a primitive node that can not be expanded further.

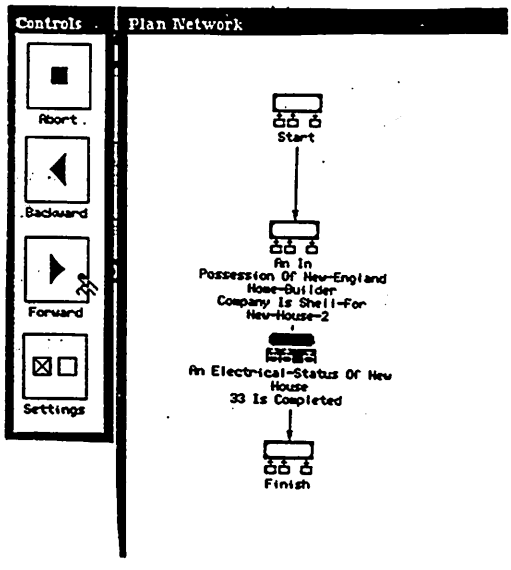


Figure 47: Graphical Plan Network with Highlighted Node.

Figure 47 shows a moderately developed network from the house construction domain. The node for electrical work is highlighted, indicating that the planner is trying to expand it.

When the planner has found an activity that satisfies the goal of the node, the graphical editor for the matching activity appears, and the flashing node is replaced by a set of nodes, corresponding to the decomposition of the activity. The decomposition can be seen in the action compartment of the editor.

The new network no longer contains the highlighted node. That node is deleted and replaced by its decomposition. The new nodes that replace the old highlighted node are highlighted by a dot pattern, to show the changes in the network to the users.

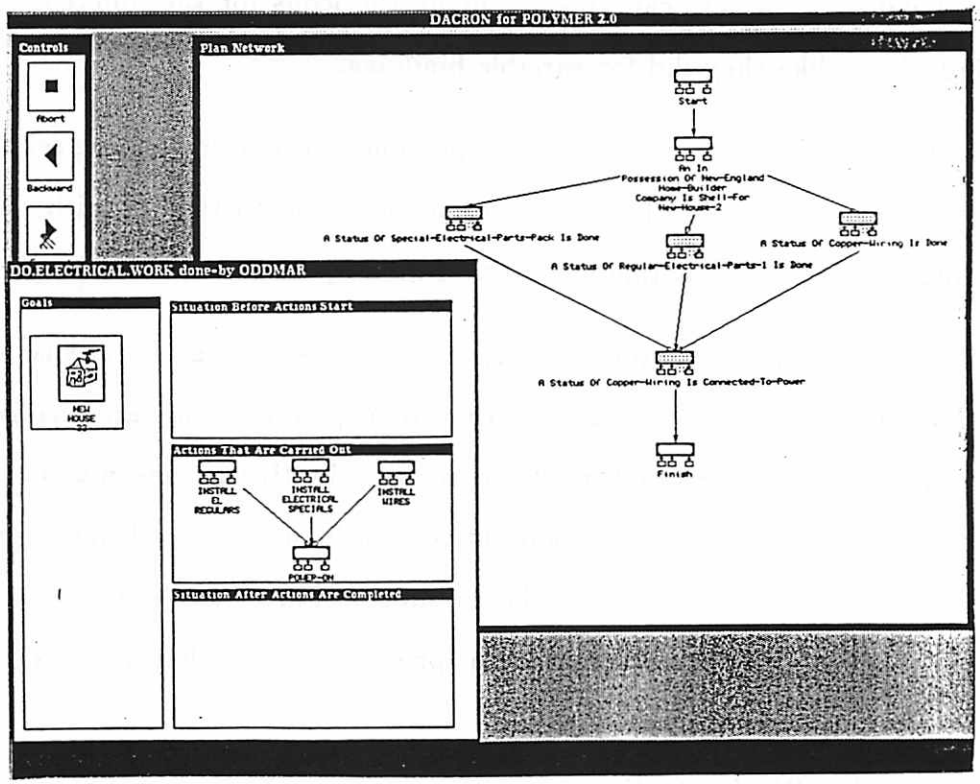


Figure 48: Refined Plan Network with Editor for Activity.

Figure 48 shows an example of an expanded network. The node for electrical work no longer exists. It is replaced by the nodes for wire installation and the nodes for appliances installation. The editor for the activity that was used to replace the node is shown on the lower left in figure 48.

If users hit the Backward button in this situation they can reobtain the network from figure 47. This allows users to see which node was expanded into which set of new nodes.

If more than one possible expansion is found for a subgoal, the users are asked to make a selection. This is done by displaying the icons for the possible plan expansions. Users can choose among the icons for the different possible activity icons like they did for variable bindings.

If more than one variable binding is possible during the planning session, the users will choose a value from an interface identical to that of initial variable bindings. This allows complete control of the objects involved in the plan.

The users can encounter three types of nodes during the advice session. **Subgoals** have been discussed above – they are expandable, and are accomplished by some plan. **Tool Executable** nodes represent primitive actions in a plan. **Agent Executable** nodes support the cooperative aspect of the knowledge acquisition and planning systems. Certain tasks, or subgoals, are 'contracted out' to other agents. Such nodes are also regarded as primitive in the plan network.

6.5.3 Setting Display Options

There are certain options and customizations available for the advice session. Users might only be interested in a linear sequence of actions, not in the higher level goals and bindings. A simplified version of the animation is produced for these users at their request. Other users might be interested in the possibility of parallel execution of certain activities, or in the constraints among agents. Choosing from a menu of display options, we satisfy various classes of information and display needs.

6.6 Summary of the Functionality of DACRON

DACRON's purpose is to acquire knowledge and to offer advice. These two top level functions are accomplished over a number of lower level functionalities. To enter knowledge, experts can specify new objects, activities and scenarios by filling out form-like editors. Dedicated editors for objects, activities and scenarios exist. Experts fill editors by typing text or by coping icons representing other units in the knowledge base. DACRON limits the range of icons and text permissible in the editors, thereby simplifying the users' task. DACRON also anticipates user actions and prepares the editors.

Experts start the specification process by finding an icon that matches the unit to be specified most closely. They open its editor, type a name for the new unit and enter values. If no icon exists that matches the users' needs, they can construct new icons by augmenting existing icons for classes. The specification process finishes by closing the editor. The new icon appears in the center of the knowledge base window.

The larger the knowledge base, the harder it becomes for users to find a desired icon to copy to an editor or to open for specification. DACRON offers a syntactic and semantic search facility that lets users quickly locate one or more icons that meet a certain criterion. The icons are presented in a new window and users can pick them and use them in the specification of the current unit. Scrolling a knowledge base window or reordering its content might also help users find an icon or clusters of icons needed in the specification of a new unit. If user are uncertain about the content of an icon, they can view it by opening its editor.

Locating desired icons is equally important for the initiation of advice, DACRON's other top level functionality. Scrolling, reordering and searching can be used to find the icon for the activity user seek advice on. During the advice process users can graphically bind values to variables, select scenarios as initial world states for the execution of an activity and select graphically among different icons, representing ways to accomplish a subgoal. Users control the advice process over a VCR-like control panel. Advice is presented as a sequence of graphical plan networks. Every new network is more detailed than the previous one. Users can move forward and backward in the sequence of networks employing the VCR control. Nodes in the network can be inspected similar to regular icons.

CHAPTER 7

EVALUATION AND USABILITY

It is necessary to state criteria for the use and the operation of systems. The functionality, correctness and efficiency of a system should be verifiable. The ease of use of interface features should be operationalized and provide behavioral measures.

Benchmarking tests and data for software products are becoming a standard [Luc71]. The measurement of usability has emerged only recently [WBH87,Gou89] but is quickly becoming an essential part of sound interface design.

In the case of knowledge acquisition and display systems, few benchmarking tests have been performed. No usability studies have been done. This is particularly surprising as knowledge acquisition systems are supposed to be tools for domain experts, not computer programmers. It was usually accepted that the tool and the interface met the design criteria, when *the experts could use it.* The advice given by the system, or the solutions proposed by it were also never strictly evaluated concerning their correctness and relevance to the user.

The *can-use-it criterion* is too vague to demonstrate that a system has met requirements nor does it allow for comparison to similar systems. That can

only be achieved by the explicit operationalization of usability goals and the measurement of system and user performance as discussed in section 2.3.5.

In addition to these usability goals, we provide statistics that enable the comparison of DACRON to other knowledge acquisition systems. These statistics can be viewed as parameters that occurred during the evaluation of the usability goals. The statistics provided involve data that is independent of the user system interaction. Examples are the frequency of invocation of the consistency checker and the average number of acquired units.

7.1 Deriving Usability Criteria

To evaluate the DACRON system, we have to solve a number of conceptual problems not usually addressed in benchmarking or usability engineering. The first problem consists of the distinction between low level usability goals, called *micro goals*, and usability involving the overall functionality and correctness of the system, called *macro goals*. Traditionally, usability engineering has addressed the micro goals, such as *moving an icon, opening a window, finding a menu item, etc.* and benchmarking is concerned with systems performance, e.g. *processing time, memory space, turnaround time, etc.* The systems performance, in our case the correctness and completeness of the specified knowledge base and the quality of advice given, is dependent on higher level functionality like specifying a correct decomposition, creating correct object classes or binding variables to values during the advice session. We therefore combine these two issues, systems performance and higher level functionality, into the macro usability goals of the system.

Another issue concerning the evaluation of DACRON is the uniqueness of the system. There are no previous systems to compare to DACRON, nor is there a way to simulate DACRON's functionality without a computer system. These facts make it very hard to set and plan for distinct performance values.

We try to deal with this problem by outlining the important issues the DACRON system set out to tackle and translating these issues into measurable tasks. The four major issues in DACRON are:

1. Can DACRON be used to specify a task domain?
2. Does DACRON elicit relevant knowledge from domain experts?
3. Does the acquired knowledge make sense to users seeking advice?
4. Is the interface and the language easy to manipulate?

We tackle the first issue in the first experiment (section 7.2), where graduate students used DACRON to specify activities that were textually presented (see appendix B). The second issue was explored in a second experiment (section 7.3). Domain experts specified parts of their domain knowledge. This experiment also touches on the first issue because specification follows elicitation. The third issue is addressed in the third experiment, where graduate students and domain experts were presented with advice acquired in the previous experiments (section 7.4). The interface and language issues were part of all three experiments. The results are reviewed in section 7.5.

Usability engineering is usually restricted to a single user. In the case of DACRON we are concerned with multiple users, possibly from the same domain. Individuals in the domain have different tasks and responsibilities. To correctly capture the knowledge about a domain, DACRON must not only be usable for

a single person, but it must facilitate the interaction and links between the knowledge of multiple users. We call this issue the *group usability problem.* We propose techniques to deal with this problem in section 7.1.1. We expect this problem to grow in importance, as the field of computer supported cooperative work receives more attention.

The DACRON system was developed with continuous user involvement. Many small design decisions were made during that process. The planned performance levels for the usability goals were set by the best performance of the most experienced users during that period. We planned to make it possible for all users to perform at that level. The evaluation reported in this chapter concerns the final version of DACRON and the performance of all its users.

7.1.1 Usability Criteria for Knowledge Acquisition

The goal in building the DACRON system was to provide a system for people who are experts in their respective domain but not familiar with knowledge engineering, planning or programming. DACRON's intention was to allow these people to specify knowledge about the tasks they perform and the objects they encounter directly.

The question of group usability is addressed by the compatibility of overlapping knowledge. Experts starting to enter knowledge will expect certain pieces of knowledge to be there. These pieces of knowledge could have been entered previously by other experts. Group usability regards the problem of experts finding and identifying this previous knowledge. This problem is particularly interesting in hierarchical organizations where experts might want to link their

knowledge to the knowledge or their supervisors or vice versa when supervisors want to express activities in terms of knowledge from their subordinates.

The macro goals for knowledge specification, their definition, operationalization and measure are:

- Start New Plan: The time to find the icon for specifying a new plan, opening it, and typing the name for the new plan in the editor;
- Update Plan: The time it takes the user to find the icon for the desired plan, opening it and starting to make the desired changes;
- Specialize Plan: The time it takes the user to find the icon closest to the new plan, starting a new editor from that icon, naming it and starting to add additional knowledge;
- Goal Specification: The number of icons selected to specify a goal wff in a plan;
- Precondition Specification: The number of icons selected to specify a precondition wff in a plan;
- Effect Specification: The number of icons selected to specify an effects wff in a plan;
- Subgoal Specification: The number of icons selected to specify a subgoal wff;
- Subtask Specification: The number of icons selected to specify a subgoal, when the desired subgoal is the goal of an existing plan;
- Linking Upwards: Existing activity icons that correspond to the subgoal are used in the specification; these icons come into existence from previous specifications by other domain experts and therefore represent a form of knowledge interaction in a group; measured by the percentage of successful interaction;
- Linking Downwards: Plans, consisting only of goal, which are left behind from a different domain expert are updated these plans form a link between experts with different roles in an organization; measured by the percentage of successful updating;

- **Rationale Specification:** The ease of indication whether two icons in the decomposition compartment are sequential or not, measured by the number of wrong mouse clicks divided by total mouse clicks to position the indication arrows;
- **Fill Out Object:** The ease of filling out a form for an object, measured by the time to find the appropriate form, opening the icon for it and typing the name for the new instance;
- **Update Object:** The ease of updating an existing form describing an object, measured by the time to find the desired form and opening the icon;
- **Set Row Value:** The effort it takes a user to enter values in the rows of a form, measured by the number of wrong value selection;
- **Create New Form:** The ease of creating a previously not existing form, measured by the time to find the closest related form, opening it and naming it;
- **Create New Row:** The effort it takes to create a new row in a new form, measured by the number of wrong selections in a menu;
- **Specify Valueclass:** Concerns the usability of the feature for strong typing of values, measured by the number of wrong icon selections.

The temporal measurements suggested, include the time to plan an action as well as its execution. We start measuring from the time when a subject expresses the intention to interact, e.g. find an icon or type a value, until that action is completed.

The second question concerning the acquisition of knowledge, whether DACRON can elicit relevant knowledge, can be broken down into the following macro goals with their respective operationalization and measure:

- **Decomposition Trigger:** A given goal results in the specification of subgoals; measured as ratio of subgoals to goals;

- **Decision on Primitives:** New plans, consisting only of a goal, are created when no activity icon corresponding to a subgoal exists. These icons are left in the knowledge base for the appropriate domain expert to update them;
- **Object Creation:** Experts create forms for objects that are needed in a plan; ranked by knowledge engineer;
- **Property Creation:** Newly created forms posses sufficiently many rows and are amended as need in the wff specification; ranked by knowledge engineer;
- **Instance Creation:** Experts create instances of the forms when they are needed;
- **Completeness:** Sufficient knowledge to give advice is collected in a session; ranked by another expert;

7.1.2 Usability Criteria for Knowledge Display and Advice

This section defines the macro goals for the display and advice case. These goals concern the the functionality involved when novices want to learn about a task or when the system is run to simulate the execution of a task as described in section 6.5.

In order to obtain measurable criteria, we operationalized the usability goals. The operationalization of some goals was straightforward, such as counting the number of tries needed to pick the correct plan when a choice is offered. Other goals were more elusive, in particular those goals concerned with the problem solving functionality of the system such as *users understand graphical plan network* or *users identify the correct plan used for the expansion.* To operationalize

these goals, we devised a questionnaire concerning details of the process given in the experiment. Solving the questions in the questionnaire was only possible when the concept tested for was completely understood.

The above efforts resulting in the following set of usability goals, operationalizations and measurements:

- Start Advice Session: It should be easy to find the icon that corresponds to the task users want advice about; the advice process should be initiated without trouble; measured by the time it takes the user from starting to look for advice until the system start to generate it;
- Bind Variables: Variables that are used in the plans needed to generate the desired advice, should be bindable in an effortless way with correct results regarding the bound values; measured by the time it takes the user to start the binding process until the variable is bound and the number of corrective actions necessary;
- Read Network: Users can understand the graphical plan network and answer questions about the task immediately; measured by the percentage of correct answers;
- Read Expansion: Users can easily identify the node that was expanded and the nodes that resulted from the expansion; measured by the percentage of correct answers;
- Initiate Next Step: The next expansion of the plan net can be initiated in an uncomplicated way; measured by the number of wrong initiation actions;
- Review Previous Network: Once the new, expanded new is present, users can still review the previous net in a quick fashion; measured by the number of wrong interface actions;
- Identify Node Types: Users can quickly and correctly identify the type of a node in the network, they can distinguish goal nodes from tool nodes; measured by the number of correct answers on the questionnaire;
- Identify Goal of Node: The goal of a node is immediately clear to the user; measured by the number of correct answers;

- **Select among Alternative Plans:** The system allows user to correctly choose among alternative plans that can possibly satisfy the current goal node; measured by the number of wrong selections users make when picking the plan;
- **Identify New Nodes:** New nodes in the most current version of the plan network can be easily identified by the users; measured by the number of correct answers to the questionnaire;
- **Identify Expanded Node:** The node that was replaced by the new nodes can be correctly indicated; measured by percentage of correct indications;
- **Identify Plan for Expansion:** The plan used to expand the goal node can be correctly identified by the users; measured by percentage of correct answers;
- **Identify Tool Action:** Primitive nodes in the net that correspond to a tool executing a primitive task are correctly identified; measured by the percentage of correct answers;
- **Identify Agent Action:** Primitive nodes in the net that correspond to an agent executing a primitive task are correctly identified; measured by the percentage of correct answers;
- **Learning Rate:** The systems is perspicuous and users quickly gain experience in using it; measured by the ratio between cumulative interaction times in the first half of the experiment and the second half.

7.1.3 Usability Criteria for Interface Interaction

Micro usability goals concern the phenomena usually evaluated in usability engineering. The manipulation of interface entities such as icons, windows, scroll bars, etc. on a generic level fall in this category.

Micro usability goals are part of all other macro usability goals. To specify a wff for instance, it is necessary to click on an icon, make a menu selection,

position a crosshair, click the mouse button, type in text, make a second menu selection and eventually copy a second icon. High usability criteria for the micro goals therefore constitute the basis for the macro goals.

The micro usability goals, their operationalizations and measures are:

- Positioning: User can easily move the mouse on the desired positions, measured by the number of correction movements necessary;
- Indicating: The desired entity can easily by indicated, by pointing, clicking, etc, measured in the number corrective trials;
- Moving: Users can easily move icons inside a window;
- Copying: Users can easily copy icons between windows;
- Menu Access: The menu attached to an icon is easily accessible to users, measured by the number of mouse clicks it takes a user to open it;
- Menu Selection: Items in a menu, on an icon or otherwise, can be selected without difficulty, measured by the number of wrong selections;
- Naming: New entities, like editors, icons or windows, can be named quickly, measured by the time from the appearance of the new entity to the completions of the naming process;
- Repositioning: Windows can be moved effortlessly, measured by the time it takes to start the move until it is completed;
- Reshaping: Windows can be reshaped quickly, measured by the time it takes to achieve the new shape;
- Scrolling: Users can scroll quickly in a window to find a desired icon, measured by the time from starting to scroll until the desired icon is found;
- Reordering: Users can quickly reorganize the icon pattern in a window, measured by the time it takes the user from starting to give the change commands until reorganization starts;

7.2 Usability Study 1: Knowledge Acquisition in the Laboratory

The experiment reported in this study was designed to answer the question about the specification aspects in DACRON. Once experts knew what they wanted to say, were they able to express it in DACRON in a quick and correct way? We isolated the questions of knowledge specification from knowledge elicitation to identify DACRON's performance in each of these subprocesses of knowledge acquisition.

7.2.1 Experimental Setting for Knowledge Specification

The subjects in this experiment were eighteen graduate students from various departments of the University of Massachusetts. Each subject was given an individual introduction to DACRON. This introduction included the specification of plans and objects. Towards the end of the introduction, the experimenter would give more and more control to the subject. When the subjects indicated that they felt comfortable with the system, the experiment began.

Subjects were presented with a job description of either a carpenter, an electrician or a plumber (appendix B shows the job description for the carpenter). The carpenter was building a house, did the wood work, and needed to subcontract the electrician and the plumber. So three subjects formed one team. Varying the order of roles (carpenter first, then plumber and electrician or elec-

trician and plumber then carpenter) subjects found the knowledge base in one of three states (empty knowledge base, knowledge base with carpenter knowledge, knowledge base with electrician/plumber knowledge). This allowed us to infer how lower level knowledge is linked to higher level knowledge and vice versa.

We videotaped subjects and screens during the specification process. The knowledge specified by the subjects was saved in a separate knowledge base. A copy of that knowledge base was given to the next member of the three subject team in the next session. The sessions lasted an average of 75 minutes, 35 minutes introduction and forty minutes experimental specification by the subject.

7.2.2 Knowledge Specification Results

The knowledge bases produced by the subjects were compared to sample solutions for the corresponding case. Differences in naming objects were accepted as long as they were synonyms. We also accepted additional knowledge provided by the subjects that fit the theme of the domain in the experiment.

The performance of every subject was evaluated according to the usability goals from section 7.1. The measures for frequent user actions, e.g. specifying a goal, were first averaged for every subject and the averaged for all subjects. The results can be found in table 7.2.2.

All usability goals concerning knowledge specification were met. Users can specify activities with all their components. Even the effects clause can be specified correctly using DACRON. Objects are created without effort. Both classes and instances can be generated quickly.

The group usability problem is solved partially through the cooperative specification of activities. Experts use activities other experts have specified to specify their own subgoals. The reverse case is also true. Activities that consist only of a goal and came into existence through an expert specifying a wff for a subgoal but not following through on the resulting new activity, are picked up by specialists for that activity. The specialist completes the unfinished activity. We successfully achieve asynchronous cooperation through shared data structures.

7.3 Usability Study 2: Knowledge Acquisition in the Field

Having answered the question about DACRON's ability to let experts specify knowledge, we turn to the question of knowledge elicitation. Would real world experts be able to use DACRON to enter their knowledge? What kind of difficulties would they have? What structure and quality would the specified knowledge have? To answer these questions we conducted the second experiment.

Table 4: Planned and Observed User Performance for Knowledge Specification.

GOAL	MEASURE	PLANNED	OBSERVED
Start New Plan	temporal	15"	10"
Update Plan	temporal	1'	1'15"
Specialize Plan	temporal	2'	1'47"
Goal Specification	number of selections	2	2
Preconditions Specification	number of selections	2	2
Effect Specification	number of selections	2	2
Subgoal Specification	number of selections	3	3
Subtask Specification	number of Selections	1	1
Linking Upwards	% occurrence	80 %	100 %
Linking Downwards	% occurrence	80 %	100 %
Rationale Specification	ratio of wrong clicks	0.25	0.00
Fill Out Object	temporal	2'	52"
Update Object	temporal	2'	1'03"
Set Row Value	number of wrong selections	1	0
Create New Form	temporal	2'	1'21"
Create New Row	number of wrong selections	1	0
Specify Valueclass	number of wrong selections	1	0

7.3.1 Experimental Setting for Knowledge Elicitation

The subjects in this experiment were seven domain experts. Three came from the area of energy auditing. These engineers usually reviewed and calculated the gas and electricity consumption for certain buildings. Three experts were architects or architectural students, involved in the design of buildings. One expert came from the domain of technical writing, working on document generation and manual preparation.

Each subject was given the same forty minute introduction as the subjects in the first study. Every session started with an empty knowledge base. The

experimenter and the subject jointly specified some activities and objects from the experts domain. They would then agree on a certain part of the experts domain and the expert would start specifying this part by himself.

The experts and their screen behavior were videotaped, the generated knowledge bases saved. The sessions lasted an average of ninety minutes, forty minutes introduction and fifty minutes experiment.

7.3.2 Knowledge Elicitation Results

The results were compiled in the same way as they were for the first study. The behavior concerning goal criteria was averaged over all occurrences and all experts. Judging the correctness of the acquired knowledge was difficult, as there was no standard. Completeness of the knowledge, syntactic correctness and its use in planning and advice giving were the eventual criteria.

Table 7.3.2 summarizes the results. They are not as good as those in the specification case. There are two reasons for this effect, first, we are dealing with elicitation and specification, second, elicitation is harder then specification. Future research will have to address this problem.

7.4 Usability Study 3: Displaying Advice

To answer the question about the usability of the advice given by DACRON a third experiment was designed. This experiment tested the relevance of the acquired knowledge in two cases. First, helping a novice in the domain solve

Table 5: Planned and Observed User Performance for Knowledge Elicitation.

GOAL	MEASURE	PLANNED	OBSERVED
Decomposition Trigger	ratio	1/2	1/2.6
Decision no Primitives	level depth	3	3.6
Object Creation	rating	sufficient	slightly sufficient
Property Creation	rating	sufficient	sufficient
Instance Creation	% occurrence	80 %	98.3 %
Completeness	number of errors	0	0

problems. Second, assisting an expert in solving problems. In this case we could also use the experts' judgment on the advice given.

7.4.1 Experimental Setting for the Display of Advice

We designed the following experimental setting to assess the usability goals from section 7.1. Users were given a standardized thirty minute introduction to the DACRON system by the experimenter. They were then presented with a questionnaire concerning various aspects of an unfamiliar task. To answer the questionnaire, users had to start an advice session on that particular task, bind variables to unique values and seek continuous advice by expanding the plan network. Users also had to retrieve previous networks, reshape windows and scroll their contents to answer the questions correctly.

Our subjects consisted of two different groups: domain experts and graduate students. The first group consisted of seven domain experts in the fields of architecture (2), energy auditing (3) and technical writing (2). These experts were presented with problems in their respective fields and used knowledge bases

constructed by the other experts in the same domain employing the DACRON knowledge acquisition facility.

The second group consisted of seven graduate students, solving similar problems in the house building/carpentry domain in which none of the subjects was an expert.

We videotaped the subjects and the corresponding screens to obtain behavioral data. This allowed us to determine the duration and the number of attempts by the users to manipulate parts of the interface. The questionnaires allowed us to verify the correctness of the solutions the users produced.

Users spent an average of forty minutes to solve the tasks. The questionnaires were evaluated in conjunction with the videotapes.

7.4.2 Advice Display Results

The results of our usability study, compiled in table 7.4.2, show that end users can easily, correctly and effectively interact with the system. Users can start the advice session quickly select the initial settings for context and variables. All interface interactions, like moving icons, reshaping windows etc., can be accomplished by the users with ease.

We observe certain difficulties in finding the appropriate icon to initiate the advice session. This points to a weakness in the knowledge base navigation facility. Performance could possibly be improved if better search facilities were offered.

The system does enable users to operate immediately in domains with which they had little previous experience. Users commented that the system was pleasurable and easy to handle, the advice readily understandable and relevant to the problem at hand.

Table 6: Planned and Observed User Performance in Display Experiment.

GOAL	MEASURE	PLANNED	OBSERVED
Start Seeking Advice	temporal	1'	2'45"
Bind Variables	% correct	100 %	100 %
Read Network	% correct	80 %	92.9 %
Read Expansion	% correct	80 %	100 %
Initiate Next Step	ordinal	1	0
Review previous Net	ordinal	1	0
Identify Node Type	true/false	true	true
Identify Goal of Node	true/false	true	true
Select among Plans	% correct	80 %	85.7 %
Identify New Nodes	% correct	80 %	92.8 %
Identify Expanded Node	% correct	80 %	100 %
Identify Plan for Expansion	% correct	80 %	100 %
Identify Tool Action	% correct	80 %	100 %
Identify Agent Action	% correct	80 %	100 %
Learning Rate	1st/2nd score	2nd	2nd

7.5 Usability Study 4: Manipulation of Interface Entities

To evaluate the micro usability goals of DACRON, we reviewed the videotapes from the three experiments reported earlier. All occurrences of interface interactions were determined and measured according to the micro usability

goals. Measures were first averaged for every subject and then averaged over all 39 subjects.

The results in table 7.5 show that all micro goals are achieved. The manipulation of interface entities can be done quick, correctly and easily. The goals not met are *scrolling*, *reordering* and *repositioning*. *Reordering* and *repositioning* are only two seconds above the usability goals. We will therefore view them as acceptable.

Scrolling took subjects an average of 33 seconds longer than planned. This seemed not to influence the subjects overall performance. We assume, that we set the planned behavioral goal too high in this case.

Table 7: Planned and Observed User Performance for Micro Goals.

GOAL	MEASURE	PLANNED	OBSERVED
Positioning	% corrections	0.5	0
Indicating	% corrections	0	0
Moving	% correct	80 %	92.9 %
Copying	% correct	80 %	100 %
Menu Access	number of clicks	1	1
Menu Selection	% wrong selections	1	0.6
Naming	temporal	30"	12"
Repositioning	temporal	15"	17"
Reshaping	temporal	30"	19"
Scrolling	temporal	10"	43"
Reordering	temporal	10"	12"

7.6 Parameters concerning DACRON's Performance

During the experiments a number of parameters were gathered that are not directly related to the usability goals but describe the performance and use of certain components and facilities of DACRON. These parameters indicate the frequency of invocation of certain modules within DACRON and the amount of the acquired knowledge.

Each user specified an average of 8.96 activities. The domain experts in the elicitation experiment specified on average more activities then the participants in the specification-only experiment. The number of activities specified by domain experts was on average 17.57 and the average number of activities specified in the specification experiment was 5.61.

The same observation holds true for the specification of objects. Domain experts in the elicitation experiment specified an average of 30.28 object classes while participants in the specification experiments specified an average of 4.83 object classes.

The knowledge bases created in the specification study contained 253 units on average. Each knowledge base was created through the sequential efforts of three participants. In the elicitation study, the knowledge bases were also created by accumulation of expert knowledge. Those knowledge bases contained on average 236.33 units. The same knowledge bases were used in the experiments concerning the advice giving facility in DACRON.

DACRON guaranteed that all specified units were syntactically correct. It was impossible to specify a syntactically incorrect unit.

The consistency checker, whose task it was to guarantee the local consistency of newly specified units and elicit additional knowledge was invoked with the following frequency:

- 0% of all invocations to prohibit circular decomposition;
- 0% to prohibit contradictions;
- 0% to prohibit introduction of a new variable in the effects clause of an activity;
- 0% to prohibit an empty goal clause.
- 42.4% of all invocations concerned an empty decomposition compartment. In all cases this was intended by the user to allow cooperative specification of activities in the group.
- 21.2% of all invocations concerned an empty effects clause. Of all those cases, 75.0% were hits. Users added an effects specification as result of the detection of an empty clause;
- In 36.4% of the cases users would specify an alternative activity, if requested by the consistency checker.

7.7 Evaluation Summary

To evaluate the usability of DACRON and to verify the inference processes that lead to its design, we conduct experimental studies. Usability criteria for knowledge elicitation, knowledge specification and interface interaction are set. Subjects are given a standardized thirty minute introduction to the system. Depending on the study, subjects either enter knowledge or seek advice from the system. The sessions are videotaped and the generated knowledge is saved. These data are analyzed with respect to the usability criteria.

The studies show that all usability goals concerning the manipulation of the objects on the screen are met. Users can open icons, move editors, name new plans, reshape windows, etc. without problems. Training times as low as thirty minutes can produce these results. The advice given by the system is easily understood. Users can guide the advice system effortlessly. The desired advice is found quickly and interpreted correctly. The features and functionalities of the advice system are learned and used without problem. End user programming in the form of knowledge specification is also achieved easily in DACRON. *All* usability goals concerning this aspect are met. Users have no problem creating new plans, building instances of objects or creating new object classes. The iconic way to specify wff's makes this task extremely easy.

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH

In this chapter we review the results of our research in direct knowledge acquisition and display and the implementation of DACRON. Elaborating on the results and phenomena discovered tangentially, we propose further research projects.

8.1 Review of Results

The purpose of this dissertation was to show that domain experts can specify their activity knowledge in a goal-based way and that this knowledge can be used by a planner to generate advice for other users. We successfully built a visual language embedded in a direct manipulation interface, DACRON, that allowed experts to recall their knowledge about a domain in terms of activities, objects and scenarios, and to easily specify it. DACRON was built on a cognitive model of the experts' view of their own activities.

This design approach, using cognitive theories and guidelines in a proactive way, generating specifications from theories and performance requirements, is unique in the area of interface and software design. Though often demanded, no

system had been built in such a way. ETS [Boo84] was based on Kelly's theory of personality but was used only as an idea for knowledge elicitation, not for direct system design.

We also recognized that a demand for this type of system building exists but that no formal method has been developed to generate system and interface specifications on the basis of task analysis, system requirements and cognitive theories and guidelines.

We built a cognitive model that addressed the experts' view of their own activities. We tested this model experimentally. Our most important finding was that experts can recall activities in a goal-based way.

Drawing implications from the model we proposed an architecture and an interaction schema for a direct knowledge acquisition system that emphasized goal-based activities. Direct manipulation, iconic specification of wff's and labels chosen by the domain experts were key components of DACRON.

Testing DACRON with a variety of users revealed its usability. Knowledge could be easily specified. Relevant knowledge was elicited. Advice given by DACRON could be used to solve task relevant questions.

8.2 Future Research Directions

During this dissertation many interesting questions were discovered that did not directly involve its the goals. Still, these questions are of importance in the general context of computer science and cognitive engineering.

In particular, the issue of an integrated system that supports all stages in the knowledge acquisition cycle arises. Applications for the advice giving system, coupled with the experts' motivation to enter knowledge, could be addressed in the area of intelligent project management systems. Problems arising from conflicting knowledge entered by multiple experts, the delegation of specification tasks to other experts and the cooperation among multiple human and computer agents could be addressed.

The lack of formal, quantitative methods for cognitive engineering and the efforts to formalize requirements analysis in software engineering are strong motivations to concentrate attempts for integrating both. Adaptive interfaces and user interface management systems employing design knowledge to adapt interfaces to changing users and situations could be seen as one way to approach the problem. Another avenue could be taken by connecting this research to current trends in the analysis of requirements and automatic generation of specifications in software engineering.

8.2.1 Supporting all Stages of Knowledge Acquisition

DACRON addressed primarily the first two phases of the knowledge acquisition cycle: elicitation and specification. A major part of the work in the field of knowledge acquisition has been concerned with knowledge assimilation and testing [Lef87,Boo84].

Combining these approaches to build a system that supports the user in a consistent way in all stages of the extended knowledge acquisition cycle seems very desirable. To help users modify or augment the knowledge contained in a

knowledge base, an acquisition facility must be able to present existing knowledge to the users, ask for additional information, validate that information, assimilate it with what is already known and allow users to experiment with newly assimilated knowledge.

To allow experts to express themselves in their own terms, the internal representation of the knowledge base must be different from its presentation to the expert. A presentation and elicitation component of a knowledge acquisition system can use explicit knowledge about dialogs and information presentation to control when and how information is displayed or elicited. Context derived from the knowledge acquisition process can also be used to refine elicitation and specification processes.

Feedback from other modules in a system supporting all stages of knowledge acquisition could be used to determine what knowledge is missing or which course the elicitation dialog should take. The incongruencies detected by modules responsible for local and global consistency as well as the modules for knowledge assimilation and testing could advise the elicitation component as to what additional information is required. By understanding which knowledge is to be acquired and why, the elicitation component can adapt the representation of knowledge to the context.

In addition to checking the local consistency of acquired knowledge, a local validation module could check new knowledge for its completeness. These checks involve heuristic tests, some of which would be domain-independent and others which would require application-specific knowledge. Domain constraints and information about the datatypes or cardinality of new knowledge could provide sources for these heuristic tests. Inconsistencies or incompleteness detected

by this validation process could be used to guide the elicitation of additional knowledge.

The knowledge assimilation and global verification stages involve interactions among individual chunks of knowledge. The structure of plan knowledge improves detection of these interactions compared to rule-based systems, while it maintaining a high degree of modularity in the knowledge base. New knowledge should therefore not simply be added to the knowledge base. It should be explicitly assimilated to allow the detection of interactions or conflicts. The interactions and conflicts can guide the acquisition process in the modification of existing knowledge or the re-elicitation of conflicting knowledge.

A first step in knowledge assimilation is the comparison of new knowledge to portions of an existing knowledge base. With the growing size of knowledge bases, a focus problem arises. By anticipating modification, as in the KNAC system, knowledge acquisition systems could try to cope with this problem [Lef87].

Goal-based knowledge is acquired with the expectation that it contributes to the formation of a plan to achieve a goal in an application domain. To test the pragmatic correctness of the modified knowledge base, users should be allowed to "exercise" portions of that knowledge base. After attempting local and global validation of new knowledge, a final test is provided to view and experiment with ways in which new knowledge actually performs.

The last stage in knowledge acquisition, "exercising" the acquired knowledge in the context of the knowledge base, appears to be a hard research problem. This problem bears resemblance to the problem of program verification. Drawing on results in that field would be helpful. Considering the rate of progress in the

field of program verification one should not expect rapid development in the efforts to verify knowledge bases.

8.2.2 Formal Methods for Cognitive Systems Engineering

During this dissertation we started to develop an integrated cognitive systems engineering approach, to which we will refer as ICSE. This approach transcends the traditional cognitive engineering approach that tries to optimize designs of human computer-interaction. ICSE also goes beyond software engineering which focuses on domain analysis, system correctness and performance.

We approach system design from the human and the machine side, by considering requirements of the task, cognition and social requirements and technical ones. We try to respond to the requirements with explanatory models that allow us to draw design implications, generate initial design and optimize it employing user participation, rapid prototyping and mathematical optimization techniques. Finally ICSE integrates usability engineering and evaluation of the joint cognitive human computer-system. Developing ICSE was not a goal of this research. It emerged as a combination and integration of methods used to solve the given problems in a coherent way. It seems necessary to continue research on this holistic approach for systems design.

We recognize two immediate issues. First, the formal description of ICSE and second, the development of automatic design generation techniques.

A first approach to formalize ICSE would be the integration of findings in cognitive science into the software design process. This concerns mainly the development phase of the software life cycle. User participatory design and

usability engineering could be formally integrated using the formal approaches given in existing models of software engineering [VR84]. A first approximation of ICSE could be a simple augmentation of an existing formalism for software engineering.

We discovered a lack of formal, qualitative methods in cognitive engineering and software engineering to generate design on the basis of the requirements analysis and the implications. We also found demands in the literature to make the use of cognitive theories proactive, rather then retroactive [WW87,Bar87].

One step to generate interfaces on the basis of cognitive theories and interface design guidelines is represented in adaptive interfaces and user interface management systems. These systems are more then mere interfaces that present objects on a screen.

At the their heart is the interface controller. The interface controller manages the output requests by the application program and the inputs from various user input sources, such as keyboard, mouse, touch glove, voice, etc. To manage these requests and generate adequate interfaces, the controller draws on display and task histories and on various sources of knowledge.

The most important knowledge source contains interface design knowledge, dialog knowledge, presentation knowledge and context knowledge. These bodies of knowledge can be represented in various ways, such as rules, frames, etc. The controller uses these sources to generate a new version of the interface that is cognitively sound and bears most relevance to the current situation. Figure 49 shows the architecture of the CRUISE adaptive interface developed

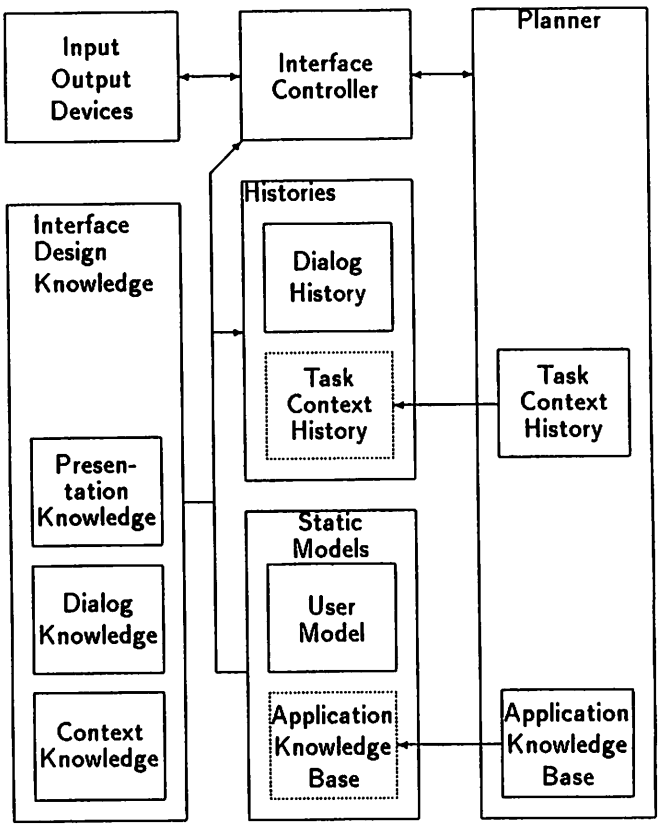


Figure 49: Architecture of Adaptable Interface.

at the Collaborative Systems Laboratory at the University of Massachusetts [ML89b].

CRUISE uses knowledge about users, task context and ways to present knowledge. Based on the current state of the screen and the dialog history CRUISE "redesigns" the interface using its explicit design knowledge. This approach could be expanded to first redesign system functionality and eventually generate initial designs for systems.

APPENDIX A

IMPLEMENTATION NOTES

DACRON was implemented on a TI Explorer 2 using the KEE knowledge engineering environment (trademark of Intellicorp). The architecture of DACRON is comprised of modified units from the KEE graphics subset. DACRON is a real-time graphical interface system, containing 10,877 lines of code.

Figure 50 shows the hierarchy of graphical editors. The hierarchy of icons is given in figure 51, being a printout of the unit specification file. The dynamic behavior and interaction functionality was defined in Common Lisp on the KEE objects. Selected functions for the creation of activities and the copying of icons are given below.

```
;;;  -*- Mode:Common-Lisp; Package:DACRON; Base:10; Fonts:(COURIER) -*-

;;; POLYMER RESEARCH GROUP
;;;       COMPUTER AND INFORMATION SCIENCE DEPARTMENT
;;;       UNIVERSITY OF MASSACHUSETTS
;;;       AMHERST, MASSACHUSETTS 01003

;;;  Copyright (C) 1989, Dirk E. Mahling,
;;;    Collaborative Systems Laboratory,
;;;    University of Massachusetts. All rights reserved.


(in-package 'dacron)
```

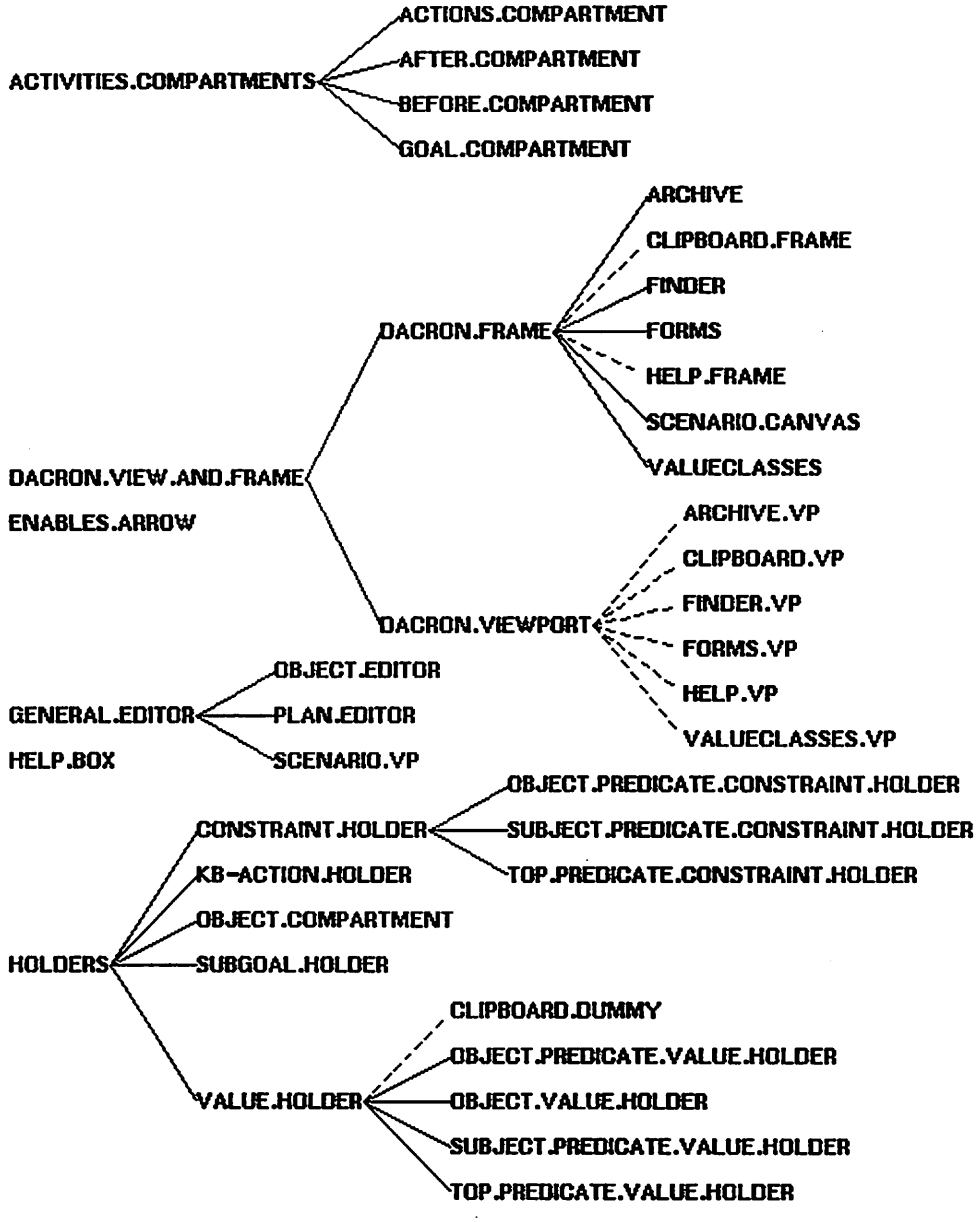


Figure 50: Graphical Hierarchy of DACRON Editors.

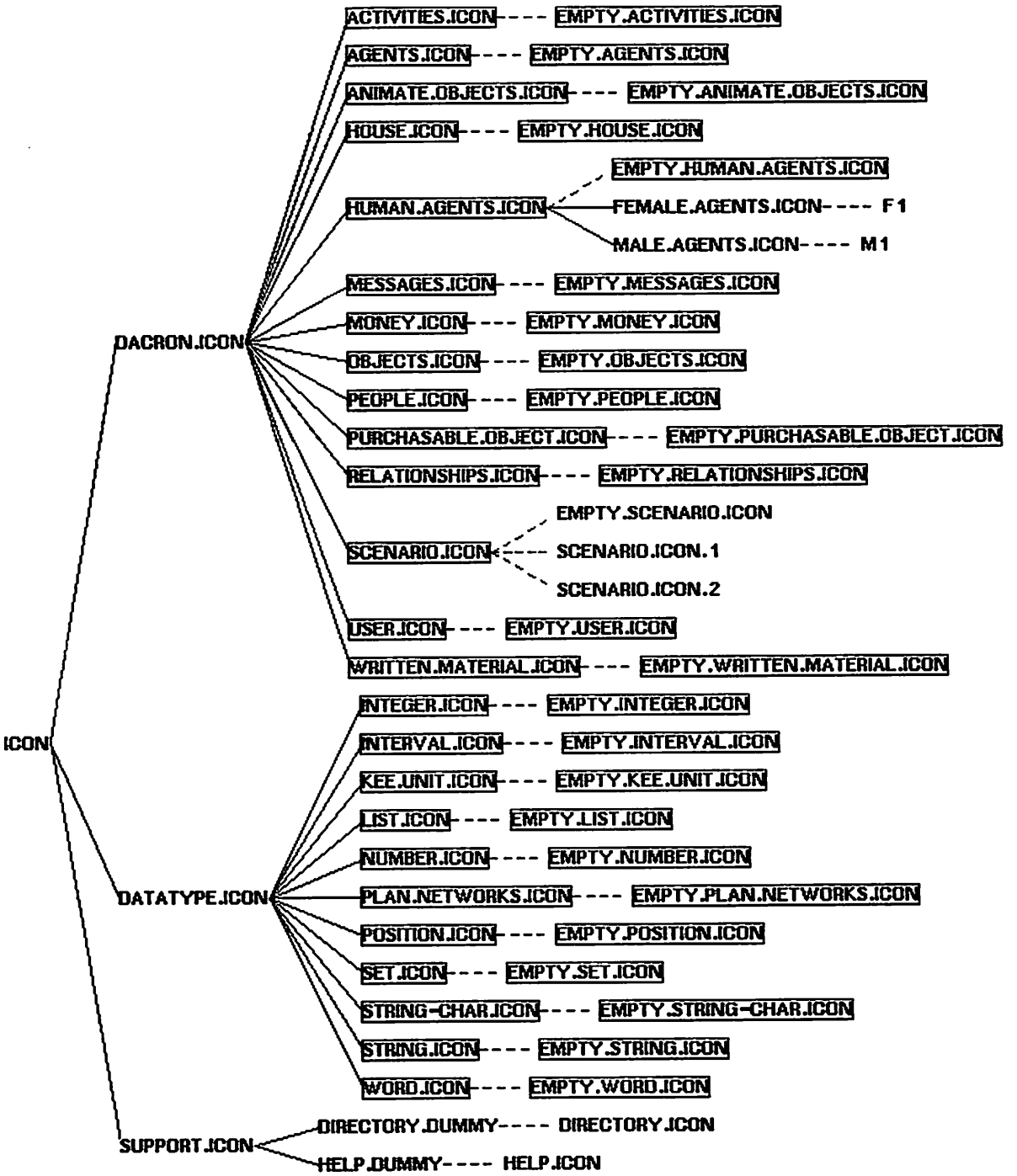


Figure 51: Graphical Hierarchy of DACRON Icons.

```
;;; top level copy function;
;;; checks if icon can  legally  be copied to the destination;
;;; then determines the type of destination:
;;; object.editor, plan.editor, scenario.editor,
;;; the finder.window or the clipboard.
;;; if none of those error message;
;;; also determines if the copied icon
;;; should represent a variable rather than a unit.
;;; depending on the case, specific copy functions take over;
;;; all clone the bitmap;



(defun copy.to.comp (self)
"takes an icon, memorizes it, changes the cursor
to copy.mode, grabs the new superpciture and position
and puts a clone of the icon right there"
(dacron.run.function
(lambda (self)
(let* ((superpic (mouse-object nil nil '"click in the
compartment you want to copy to"))
(pos (mouse-position-in-picture superpic t))
;gives position
;in compartment where
;to place the clone
(legal-comp-flag
(loop for destination
in *copy-destinations*
when (or
(unit.ancestor.p destination
superpic
'member t)
(unit.ancestor.p destination
superpic
'subclass t))
return t)))
(dacron.debug ("superpicture is ~s,
legal-flag is ~s" superpic legal-comp-flag))
(if legal-comp-flag
(let* ((vp (or (first (k:find-viewports superpic))
superpic))    ;oddmar.facet
(variable-flag (variable.p self))
; (equal 'forms.vp (unit.name selfs-vp)))
;or when from clipboard....then
(constraint-flag (unit.ancestor.p
(unit 'constraint.holder) superpic 'member))
;empty.xxx.icons are not variables
; in the constraint definition but stand for classes!
```

```
(scenario-flag (unit.descendant.p vp
(unit '(scenario.vp dacron)) 'member))
;oddmar.facet
(finder-flag (equal vp (unit 'finder.vp)))
(clipboard-flag (equal vp (unit 'clipboard.vp)))
(temp.unit (get.value vp 'attached.temp.unit)))
(cond (temp.unit
(copy.to.temp self
pos superpic
variable-flag
constraint-flag vp temp.unit))
(finder-flag
(copy.to.finder self pos superpic variable-flag))
(scenario-flag
(copy.to.scenario.vp
self pos vp superpic variable-flag))    ;oddmar.facet
(clipboard-flag
(copy.to.clipboard self pos superpic))
(t (dacron.message
"You did not start by filling out a form or
creating a new form, sorry can not copy icon."))))
(dacron.message "You can not copy to that place.
You can only copy to compartments in plan
editors, object editors and in the finder."))))
self))



;;;
;;; copies icons for the puropose of specification
;;; of units; specification
;;;         must be saved seperately except for objects
;;; that branch therefore is deepest;
;;; follow objects also in the varible branch.
;;; Old functionality is in specify-values as
;;; commit.form.specified.by.copy.to.temp
;;; 5 jul 89; copy to comp split up in the test if it is legal
;;; to clone then the big rest in copy.to.comp.really

(defun copy.to.temp (self pos superpic variable-flag
constraint-flag vp temp.unit)
(let* ((clone   ;;;to.do put a test here activities to
actions.cpt only rest.icons to rest.cpts only
(when (or (and (unit.descendant.p self
(unit 'activities.icon) 'member)
(unit.descendant.p superpic (unit 'actions.compartment) 'member))
(and (not (unit.descendant.p self (unit 'activities.icon) 'member))
(not (unit.descendant.p superpic
(unit 'actions.compartment) 'member))))
```

```
(unitmsg self 'clone! pos superpic))))  ;creates the clone
(if clone (clone.to.temp clone self pos
superpic variable-flag constraint-flag vp temp.unit)
If you tried to copy an icon for things or people etc.,
into the actions compartment of a task-editor,
use an icon form empty task and then specify the subgoal."))))



(defun clone.to.temp (clone self pos superpic variable-flag
constraint-flag vp temp.unit)
(let* (
(kb-wff-flag (get.value self 'value.of.poly-slot)))
;kb-wff copied rather than single icon
(add.parent clone 'instance.icon 'member)
;makes the clone an instance.icon type
(put.value clone 'highlightp nil)
(when (member (unit.name self) *empty-dacron-icons*)
(remove.all.values clone 'mouseleftitems))
(when (and variable-flag (not constraint-flag))
;variables only in value specs.
(copy.icon.as.a.variable self clone vp temp.unit pos superpic))
(cond ((or (unit.ancestor.p (unit 'goal.compartment) superpic 'member)
;starting a kb.wff
(unit.ancestor.p (unit 'subgoal.holder) superpic 'member)
(unit.ancestor.p (unit 'before.compartment) superpic 'member))
(add.subject.predicate.value.holder clone superpic  nil))
((unit.ancestor.p (unit 'after.compartment) superpic 'member)
;starting a kb.modification
(copy.icon.to.after.compartment clone superpic pos))
((unit.ancestor.p (unit 'actions.compartment) superpic 'member)
;start a kb.decomposition
(if variable-flag
(set.undefined.activity.step clone superpic temp.unit)
(add.new.wff.to.decomposition  temp.unit clone )))
((unit.ancestor.p
(unit 'd::object.predicate.value.holder) superpic 'member)
(commit.simple.kb-wff superpic))
((or (unit.ancestor.p (unit 'p:objects) temp.unit 'member t)
;now for objects
(unit.ancestor.p (unit 'p:objects) temp.unit 'subclass t))
(update.temp.object self superpic temp.unit)))
(after.copy.guidance superpic clone)
(when kb-wff-flag (commit.existing.kb-wff clone))
(when (unit.exists.p superpic)
(unitmsg superpic 'k:expose!))))

(defun after.copy.guidance (superpic clone)
```

```
(declare (ignore clone))
(cond ((or (unit.parent.p (unit 'after.compartment) superpic 'member)
(unit.parent.p (unit 'goal.compartment) superpic 'member)
(unit.parent.p (unit 'subgoal.holder) superpic 'member)
(unit.parent.p (unit 'before.compartment) superpic 'member))
If you do not specify values on this icon you just
copied, DACRON will forget about it.")))




;;;
;;; copying to clipboard allows to keep frequently
;;; used icons in one window, thereby eliminationg search time
;;;

(defun copy.to.clipboard (self pos superpic)
(let ((clone (unitmsg self 'clone! pos superpic)))
(add.parent clone 'proto.icon 'member)
(put.value clone 'highlightp nil)))

;;;
;;; copying to the finder.vp starts a kb-wff
;;; definition when we are in one of the plan compartments
;;; or an object specification for search that
;;; starts immediately
;;;


(defun copy.to.finder (self pos superpic variable-flag)
(let ((clone  (unitmsg self 'clone! pos superpic))
;create.the clone
(start-search-flag (unit.ancestor.p
(unit 'object.compartment) superpic 'member)))
(add.parent clone 'instance.icon 'member)
;makes the clone an instance.icon type
(put.value clone 'highlightp nil)
(when (member (unit.name self) *empty-dacron-icons*)
(remove.all.values clone 'mouseleftitems))
(if start-search-flag (search.for.component)
(when variable-flag
(copy.icon.as.a.variable self clone 'finder.vp nil pos))
(with-keeio (format t "the superpicture is ~s" superpic))
(cond
((or (unit.ancestor.p (unit 'goal.compartment) superpic 'member)
;starting a kb.wff
(unit.ancestor.p (unit 'subgoal.holder) superpic 'member)
(unit.ancestor.p (unit 'before.compartment) superpic 'member))
(add.subject.predicate.value.holder clone superpic  nil))
```

```
((unit.ancestor.p (unit 'after.compartment) superpic 'member)
;starting a kb.modification
(copy.icon.to.after.compartment clone superpic pos))))))


(defun copy.icon.to.after.compartment (clone superpic pos)
(let ((kb-action-holder (unitmsg 'kb-action.holder 'k:create.instance!
'd:dacron nil superpic
pos nil *kb-action-holder-height*
*kb-action-holder-width*
nil t 'k:left 'k:top *kb-action-holder-font* 1))
(kb-action))
(add.subject.predicate.value.holder clone kb-action-holder nil)
(setq kb-action (pop.up.choice.menu
'"What do you want to do with situation after?" '(Add Set Delete Help)))
(if (equal kb-action 'Help)(dacron.message
"The effect of the actions you are starting
to specify can either add a new fact to the
current state of affairs [ADD], it can remove
something that was true [DELETE] or the
effects can reset something that is
no longer true in the old way [SET].")
(put.value kb-action-holder 'strings (list (string kb-action))))))



(defun copy.icon.as.a.variable (original-icon icon
destination-vp temp-unit position &optional superpic)
"copying an icon from the forms vp means that the whole
class can be used, therefore this copied clone represents a VARIABLE
to POLYMER. We need the name for the variable, put CONSTRAINTS
on the variable, set the attached.unit value of the icon."
(declare (ignore destination-vp position))
(let* ((variable-name (get.variable.name icon))
(icon-parents-unit (get.value original-icon 'attached.unit))
(activity-flag  (when (or (unit.ancestor.p
(unit 'p:activities) icon-parents-unit 'member t)
(unit.ancestor.p (unit 'p:activities)
icon-parents-unit 'subclass t)) t))
(variable-symbol (if activity-flag (intern variable-name 'p)
(intern (format nil "?~A" variable-name) 'p))))
(put.value icon 'label (string variable-symbol))
(when (and temp-unit (not (or (unit.ancestor.p (unit 'p:activities)
icon-parents-unit 'member t)
(unit.ancestor.p
(unit 'p:activities) icon-parents-unit 'subclass t))))
(add.value temp-unit 'p:constraints (list 'member variable-symbol
```

```
icon-parents-unit)))
(put.value icon 'variable.name variable-symbol)
(put.value icon 'attached.unit icon-parents-unit)
(when (or (unit.ancestor.p (unit 'p:objects) temp-unit 'member t)
;now for objects
(unit.ancestor.p (unit 'p:objects) temp-unit 'subclass t))
(update.temp.object icon superpic temp-unit)
(dacron.debug ("also in icon.as.var")))))



(defun get.variable.name (superpicture)
"same as get.step.name or get.a.name, slightly adapted for variables"
(let* ((prompt   (k:box-string-create-instance
(unit 'k:(box.strings keepictures))
'dacron nil superpicture
'(-200 0) nil 400 30 '("type name for variable in: ")
nil nil nil nil 2))
(typed-in-string))
(unitmsg prompt 'type.in!)
(setq typed-in-string (first (get.value prompt 'strings)))
(unitmsg prompt 'delete!)
(string-upcase typed-in-string)))


;;; allows to copy instance icons within and among editors
;;; if with-values is true the whole kb-wff is copied;

(defun copy.instance.icon (icon with-values)
(let* ((superpic (get.value icon 'superpicture))
(valid-flag (unless
(or (unit.ancestor.p
(unit 'object.predicate.constraint.holder)  superpic  'member)
(unit.ancestor.p
(unit 'object.predicate.value.holder)  superpic  'member)
(unit.ancestor.p (unit 'actions.compartment)  superpic  'member)
(unit.ancestor.p (unit 'object.compartment)  superpic  'member)
(unit.ancestor.p (unit 'object.value.holder)  superpic  'member)) t)))
(when valid-flag
(if with-values
(copy.to.comp icon)
(clone.instance.icon icon)))))

;;; clones an instance icon in an editor to another
;;; compartment in the same or another editor, clones
;;; just the icon not the values on it

(defun clone.instance.icon (from-icon)
```

```
(dacron.run.function
(lambda (self)
(let* ((superpic (mouse-object nil nil
'"click in the comparment you want to copy to"))
(pos (mouse-position-in-picture superpic t))
;gives position in comparatment where to place the clone
(legal-comp-flag  (loop for destination in *copy-destinations*
when (or (unit.ancestor.p destination superpic 'member t)
(unit.ancestor.p destination superpic 'subclass t))
return t)))
(dacron.debug
("superpipcture is ~s, legal-flag is ~s" superpic legal-comp-flag))
(if legal-comp-flag
(let ((clone  (unitmsg self 'clone! pos superpic 'd::dacron nil 0)))
(add.parent clone 'instance.icon 'member)
(put.values clone 'value.of.poly-slot nil)
(cond ((or (unit.ancestor.p (unit 'goal.compartment) superpic 'member)
;starting a kb.wff
(unit.ancestor.p (unit 'subgoal.holder) superpic 'member)
(unit.ancestor.p (unit 'before.compartment) superpic 'member))
(add.subject.predicate.value.holder clone superpic  nil))
((unit.ancestor.p (unit 'after.compartment) superpic 'member)
;starting a kb.modification
(copy.icon.to.after.compartment clone superpic pos))
((unit.ancestor.p
(unit 'd::object.predicate.value.holder) superpic 'member)
(commit.simple.kb-wff superpic))))
(dacron.message "You can not copy to that place.
You can only copy to compartments in plan editors, object
editors and in the finder."))))
from-icon))
```

APPENDIX B

INSTRUCTIONS FOR SPECIFICATION EXPERIMENT

The following instructions where given to subjects who played the carpenter role in the knowledge specification study:

> Assume you are a carpenter and builder with the New England Home Builder Company. You build houses and sell them. Because you are not very efficient with wiring and plumbing you always subcontract a plumber and an electrician do to those jobs.
>
> To increase the efficiency of your business you have decided to use DACRON for project management purposes. Now you have to tell DACRON what it actually is that you are doing.
>
> You build houses by erecting a frame, subcontracting an electrician, and subcontracting a plumber. The plumber and the electrician can only start working when you are done with the frame. They have to install the wiring and the plumbing. You don't care how the electrical wiring and the plumbing is put in. Your subcontractors will do that.
>
> To erect the frame you have to post beams and attach sheetrock. You have to do these two steps in exactly that order, because the sheetrock is attached to the beams.

Make sure you have enough beams and sheetrock before you start! Also remember that they are consumed in the activity of erecting the frame.

Thank you very much for your participation.

BIBLIOGRAPHY

[AB73] Anderson, J.R. and Bower, G.H. *Human Associative Memory.* Winston and Sons, 1973.

[AB88] Abrett, H. and Burstein, M.H. The KREME knowledge editing environment. In J. Boose and B. Gaines, editors, *Knowledge Acquisition Tools for Expert Systems.* Academic Press, 1988.

[ACM81] ACM SIGMOD. *Proceedings of the workshop on data abstraction, databases and conceptual modelling*, 1981.

[ACS89] Angelaccio, M., Catarci, T. and Santucci, G. QBD: A fully visual system for E-R oriented databases. In *1989 IEEE Workshop on Visual Programming Languages.* IEEE, 1989.

[And83] Anderson, J.R. *Architecture of Cognition.* Harvard University Press, 1983.

[AT64] Adam, C. and Tannery, P., editors. *Descartes: Oeuvres.* Bibliotheque des textes philosophiques, 1964.

[Bae81] Baecker, R. Sorting out sorting. SIGGRAPH, 1981. 16mm sound film.

[Bar83] Barber, G. Supporting organizational problem solving with a work station. *Transactions of Office Information Systems*, 1(1):45–67, 1983.

[Bar87] Barnard, P.J. Cognitive resources and the learning of human-computer dialogs. In J. Carroll, editor, *Interfacing Thought.* MIT Press, 1987.

[BB88] Boose, J.H. and Bradshaw J.M. Expertise transer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems. In J. Boose and B. Gaines, editors, *Knowledge Acquisition Tools for Expert Systems.* Academic Press, 1988.

[BBB+83] Buchanan, B.G., Barstow, D., Bechtal, R., Benett, J., Clancey, W., Kulikowksi, C., Mitchell, T. and Waterman, D.A. Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, editors, *Building expert systems*, pages 126 –167. Addison-Wesley, Reading, Massachusetts, 1983.

[BBT79] Bower, G., Black, J. and Turner, T. Scripts in memory for text. *Cognitive Psychology*, 11, 1979.

[BC88] Broverman, C.A. and Croft, B.C. Plausible explanation to cope with unanticipated behavior in planning. Technical Report 88-56, University of Massachusetts, 1988.

[BG86] Boose, J.H. and Gaines, B.R., editors. *Proceedings of the knowledge acquisition for knowledge-based systems workshop*, Banff, Alberta, Canada, 1986. American Association for Artificial Intelligence.

[BG87] Boose, J.H. and Gaines, B.R., editors. *Proceedings of the 2nd knowledge acquisition for knowledge-based systems workshop*, Banff, Alberta, Canada, 1987. American Association for Artificial Intelligence.

[BNC86] Baeckmann, L., Nillson, L.G. and Chalom, D. New evidence on the encoding of action events. *Memory and Cognition*, 14(4):339–346, 1986.

[Boe80] Boesch, E. *Kultur und Handlung.* Huber, Bern, 1980.

[Boo84] Boose, J. Personal construct theory and the transfer of human expertise. In *Proceedings of the national conference on artificial intelligence*, Austin, Texas, 1984.

[Bor79] Borning, A.H. *ThingLab - A constraint oriented simulation laboratory.* PhD thesis, Stanford University, 1979.

[Bra86] Bradley, P.A. Constructing a user interface with a rule based user interface management systems. In *Proceedings of the 1986 AAAI conference*, 1986.

[Bro86] Brown, M. *Human Computer Interface Design Guidelines.* Ablex, 1986.

[BS84] Brown, M.H. and Sedgewick, R. A system for algorithm animation. *Computer Graphics*, 18(3), 1984.

[CAC] CACI. SIMSCRIPT. SIMSCRIPT is a trademark and product of CACI products company.

[CGP89] Cox, T.T., Giles, F.R. and Pietrzkowsky, T. Prograph: A step towards liberating programming from textual conditioning. In *1989 IEEE Workshop on Visual Programming Languages.* IEEE, 1989.

[CL84] Croft, W.B. and Lefkowitz, L. Task support in an office system. *Transactions of Office Information Systems*, 2(3), 1984.

[CL87] Croft, W.B. and Lefkowitz, L. A goal-based representation of office work. In *Proceedings of the IFIP workshop on office knowledge*, Toronto, 1987. North Holland.

[CL88] Croft, W.B. and Lefkowitz, L. Knowledge-base support of cooperative activities. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 599 –605. Morgan Kaufman, 1988.

[Cla85] Clarisse, O.B. *Visual programming with icons.* PhD thesis, Illinois Institute of Technology, 1985.

[CMN83] Card, S., Moran, T. and Newell, A. *The Psychology of Human Computer Interaction.* Lawrence Erlbaum, 1983.

[CRM80] Charniak, E., Riesbeck, C. and McDermott, D. *Artificial Intelligence Programming.* Lawrence Erlbaum, Hillsdale, 1980.

[Don78] Donelson, W. Spatial management of information. *Computer Graphics*, 12(3), 1978.

[Dud89] Dudley, T. A visual interface to a conceptual data modelling tool. In *1989 IEEE Workshop on Visual Programming Languages.* IEEE, 1989.

[Fei84] Feigenbaum, E.A. Knowledge engineering: The applied side of artificial intelligence. *Annals of the New York Academy of Sciences*, pages 91–107, 1984.

[FGHW88] Flores, F., Graves, M., Hartfield, B. and Winograd, T. Computer systems and the design of organizational interaction. *Transactions of Office Information Systems*, 6(2), 1988.

[Fis88] Fisher, D.L. Optimization in cognitive engineering. *Lecture Notes for Cognitive Engineering Seminar, IEOR 689*, May 1988. Department of Industrial Engineering, University of Massachusetts.

[FN71] Fikes, R.E. and Nillson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 3, 1971.

[FP62] Fitts, P.M. and Posner, M.I. *Human Performance.* Brooks Cole Publ. Co., Monterey, CA., 1962.

[FS85] Frese, M. and Sabini, J., editors. *Goal Directed Behavior: The Concept of Action in Psychology.* Lawrence Erlbaum Associates, 1985.

[FWC82] Foley, F., Wallace, V., and Chan, P. The human factors of graphical interaction: Tasks and techniques. *IEEE Graphics*, 1982.

[GB88] Gaines, B.R. and Boose J.H. Knowledge acquisition tools for expert systems. In B.R. Gaines and J.H. Boose, editors, *Knowledge Acquisition Tools for Expert Systems*, pages xiii–xvi. Academic Press, 1988.

[GF84] Gould, L. and Finzer, W. Programming by rehearsal. Technical Report scl-84-1, Xerox Parc, 1984.

[Gol84] Goldberg, A. *Smalltalk80: The interactive programming environment.* Addison Wesley, 1984.

[Gou89] Gould, G. How to design usable systems. In M. Helander, editor, *Handbook of human computer interaction.* North Holland, 1989.

[GP84] Green. T.R.G. and Payne, S.J. Organization and learnability in computer languages. *International Journal of Man-Machine Studies*, 21:7–18, 1984.

[GT84] Glinert, E. and Tanimoto, S. PICT: An interactive graphical programming environment. *IEEE Computer*, 17(11):7–25, 1984.

[Gut87] Gutfreund, S. Maniplicons in Thinkertoy. Technical Report 87-39, University of Massachusetts, 1987.

[Har85] Hartson, R. *Advances in Human Computer Interaction.* Ablex, 1985.

[Hei27] Heidegger, M. Sein und Zeit. *Jahrbuch fuer Philosophie und Phaenomenologische Forschung*, 8, 1927.

[HHKW77] Hammer, M., Howe, G., Kruskal, V. and Wladawsky, I. A very high level programming language for data processing applications. *Communications of the ACM*, 20(11), 1977.

[HHW74] Hammer, M., Howe, G. and Wladawsky, I. An interactive business definition system. *SIGPLAN*, 9(4), 1974.

[HIY+87] Hirakawa, M., Iwata, S., Yosimoto, I., Tanaka, M. and Ichikawa, T. An environment for hi-visual iconic programming. In *1987 IEEE Workshop on Visual Programming Languages*. IEEE, 1987.

[HL87] Huff, K.E. and Lesser, V.R. A plan-based intelligent assistant for the process of programming. Technical report, University of Massachusetts, 1987.

[HW83] Hollnagel, E. and Woods, D.D. Cognitive engineering: New wine in new bottles. *International Journal of Man-Machine Studies*, 18(6), 1983.

[HYTI89] Hirakawa, M., Yoshimi, I., Tanaka, M. and Ichikawa, T. A generic model for constructing visual programming languages. In *1989 IEEE Workshop on Visual Programming Languages*. IEEE, 1989.

[Jac85] Jacob, R. A state transition diagram language for visual programming. *IEEE Computer*, 18(8), 1985.

[JN87] John, B. and Newell, A. Predicting the time to recall computer command abbreviations. In *CHI+GI '87 Proceedings*, 1987.

[KB72] Klaus, G. and Buhr, M. *Philosophisches Woerterbuch*. VEB Deutscher Verlag der Wissenschaften, 1972.

[KB85] Kuhl, J. and Beckmann, J., editors. *Action Control: From Cognition to Behavior*. Springer-Verlag, 1985.

[Lak80] Lakin, F. A structure for manipulation of text-graphics objects. *Computer Graphics*, 14(6), 1980.

[Lan87] Landauer, T.K. Relations between cognitive psychology and computer system design. In J. Carroll, editor, *Interfacing Thought*. MIT Press, 1987.

[LC89] Lefkowitz, L.S. and Croft, W.B. Planning and execution of tasks in cooperative work environments. In *Proceedings of the Fifth IEEE Conference on Artificial Intelligence Applications*. IEEE, 1989.

[LD85] London, R.L. and Duisberg, R.A. Animating programs using smalltalk. *IEEE Computer*, 18(8), 1985.

[Lef87] Lefkowitz, L.S. *Knowledge Acquisition through Anticipation of Modifications*. PhD thesis, University of Massachusetts, 1987.

[Lie84] Lieberman, H. Seeing what your programs are doing. *International Journal of Man-Machine Studies*, 21, 1984.

[LN77] Lindsay, P.H. and Norman, D.A. *Human Information Processing: An Introduction to Psychology*. Academic Press, New York, 1977.

[Luc71] Lucas, H.C. Performance evaluation and monitoring. *Computing Surveys*, 3(3):79 –91, 1971.

[LW87] Lethbridge, T.C. and Ware, C. Animation using behavior functions. In *1987 IEEE Workshop on Visual Programming Languages*, pages 133–146. IEEE, 1987.

[Mac86] Mackinlay, J. Automatic design of graphical presentation. In *Proceedings of the AAAI 1986*. American Association for Artificial Intelligence, 1986.

[MC88] Mahling, D.E. and Croft, W.B. Knowledge acquisition for planners. In B. Gaines and J. Boose, editors, *Proceedings of the 3rd Knowledge Acquisition for Knowledge Based Systems Workshop*, 1988.

[MC89] Mahling, D.E. and Croft, W.B. Relating human knowledge to plans. *International Journal of Man-Machine Studies*, 31:61 – 97, 1989.

[Mea85] Mead, M. Planning for user interface standards. *SIGCHI Bulletin*, 17(2), 1985.

[Mes86] Mesrobian, E. GRAPES: A graphical programming environment for simulation. Master's thesis, UCLA, 1986.

[MFCS86] Musen, M., Fagan, L., Coombs, D. and Shortliffe, E. Using a domain model to drive an interactive knowledge editing tool. In *Proceedings of the Knowledge Acquisition for Knowledge Based Systems Workshop*, pages 33.0–33.11. American Association for Artificial Intelligence, 1986.

[ML89a] Mahling, D.E. and Lefkowitz, L.S. DACRON: A knowledge acquisition and display system. Technical Report CLS-89-3-v on VHS-NTSC Video, 30 minutes, 1989. published by the Collaborative System Laboratory, COINS, UMASS, Amherst MA 01003.

[ML89b] Mahling, D.E. and Lefkowitz, L.S. Using task context to support an adaptable interface. In G. Salvendy, editor, *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, chapter 4, pages 565–573. Elsevier, Amsterdam, 1989.

[MMW85] Marcus, S.J., McDermott, J. and Wang, T. Knowledge acquistion for constructive systems. In *Proceedings of the ninth international conference on artificial intelligence*, Los Angeles, California, 1985.

[Mon85] Monk, A. *Fundamentals of Human Computer Interaction.* Academic Press, 1985.

[Mor81] Moran, T.P. The command language grammar. *International Journal of Man-Machine Studies*, 15, 1981.

[Mye83] Myers, B. Incense: A system for displaying datastructures. *Computer Graphics*, 17(3), 1983.

[ND86] Norman, D. and Draper, S. *User Centered System Design.* Lawrence Erlbaum, 1986.

[Nil80] Nilsson, N.J. *Principles of Artificial Intelligence.* Morgan Kaufmann, Los Altos, 1980.

[Nis86] Nishida, S. Speech recognition enhancement by lip information. In *CHI'86 Human Factors in Computing Systems*, pages 198–205. ACM, ACM, 1986.

[NM83] Newman, W. and Mott, T. Officetalk Zero: An experimental office system. In P. Degano and E. Sandevall, editors, *Integrated Interactive Computer Systems.* North Holland, 1983.

[NS72] Newell, A. and Simon, H. *Human Problem Solving.* Prentice-Hall, 1972.

[PG86] Payne, S.J. and Green, T.R.G. Task action grammars: A model of the mental representation of task languages. *Human Computer Interaction*, 2(2):93–133, 1986.

[PK83] Pea, R.D. and Kurland, D.M. On the cognitive prerequisites of computer programming. Technical Report TR 18, Bank Street College, New York, 1983.

[PK85] Polson, P.G. and Kieras, D.E. A quantitative model of the learning and performance of text editing knowledge. In L. Borman and B. Curtis, editors, *Human Factors in Computing; CHI '85.* ACM, Inc., New York, 1985.

[Ras86a] Rasmussen, J. A framework for the cognitive task analysis in systems design. In E. Hollnagel, G. Mancini, and D.D. Woods, editors, *Intelligent decision support in process environments*, pages 175–210. Springer Verlag, 1986.

[Ras86b] Rasmussen, J. *Information Processing and Human-Machine Interaction.* Systems Science and Engineering. North-Holland, 1986.

[RC88] Rogers, T.R. and Cattell, R.G. Entity-relationship user interfaces. *IEEE Data Engineering*, 11(2), 1988.

[Rei84] Reisner, P. Formal grammar as a tool for analyzing ease of use: Some fundamental concepts. In J.C. Thomas and M.L. Shneider, editors, *Human Factors in Computing Systems.* Ablex Publishing Corporation, 1984.

[RGR85] Rubin, R., Golin, E. and Reiss, S. Thinkpad: A graphical system for programming by demonstration. *IEEE Software*, 2(2), 1985.

[RH84] Rubinstein, R. and Hersh, H. *The Human Factor: Designing Computer Systems for People.* Digital Press, 1984.

[RW89] Reubenstein, H.B and Waters, R.C. The requirements apprentice: An initial scenario. In *Fifth International Workshop on Software Specification and Design*, 1989. appeared as Volume 14, Number 3, May 1989, of ACM SIGSOFT Engineering Notes.

[SA77] Schank, R.C. and Abelson, R.P. *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum, 1977.

[Sac74] Sacerdoti, D.E. Planning in a hierarchy of abstractions spaces. *Artificial Intelligence*, 5, 1974.

[Sac75] Sacerdoti, D.E. A structure for plans and behavior. Technical Report 109, AI Center, SRI, 1975.

[SB89] Summersgill, R. and Browne, D.P. Human factors: Its place in system development methods. In *Fifth International WOrkshop on Software Specification and Design*, 1989. appeared as Volume 14, Number 3, May 1989, of ACM SIGSIFT Engineering Notes.

[SC88] Shneiderman, B. and Carroll, J.M. Ecological studies of professional programmers: An overview. *Communications of the ACM*, 31(11):1256 - 1259, 1988.

[Sch88] Schirmer, K. Techniken der Wissensakquisition. *Kuenstliche Intelligenz: Forschung, Entwicklung, Erfahrungen*, 4/88:68 -71, 1988.

[Shn80] Shneiderman, B. *Software Psychology - Human Factors in Computing Systems.* Gallstone, 1980.

[Sim81] Simon, H. *The Sciences of the Artificial.* MIT Press, 1981.

[Ski57] Skinner, B.F. *Verbal Behavior.* Appleton Crofts, New York, 1957.

[SM86] Smith, S. and Mosier, J. *Guidelines for Designing User Interface Software.* Mitre, 1986.

[Smi77] Smith, D.C. *A Computer Program to Model and Stimulate Creative Thought.* PhD thesis, Stanford University, 1977.

[Smi86] Smith, R.B. The Alternate Reality Kit: An animated environment for creating interactive simulations. In R. R. Korfhage, editor, *1986 IEEE Workshop on visual languages*, pages 99–106, 1986.

[Ste81] Stefik, M.J. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:141–169, 1981.

[Suc85] Suchman, L.A. Plans and situated actions: The problem of human-machine communiation. Technical Report isl-6, Xerox Corporation, 1985.

[Sus73] Sussmann, G.A. A computational model of skill acquisition. Technical Report AI-TR-297, MIT AI Lab, 1973.

[Tat76] Tate, A. Project planning using a hierarchic non-linear planner. Technical Report 25, Department of Artificial Intelligence, University of Edinburgh, 1976.

[Tho78] Thomas, A. *Einfuehrung in die Sportpsychologie.* Goettingen, Toronto, Zuerich, 1978.

[Tho84] Thomas, J., editor. *Human Factors in Computer Systems.* Ablex, 1984.

[TL82] Tschiritzis, D. and Lochowsky, F. *Data Models.* Prentice Hall, 1982.

[Tuf83] Tufte, E. *The Visual Display of Quantative Information.* Graphics Press, 1983.

[Ver83] Vere, S. Planning in time: Windows and durations for activities and goals. *IEEE transactions on pattern analysis and machine intelligence*, 5, 1983.

[VR84] Vick, C.R. and Ramamoorthy, C.V., editors. *Handbook of Software Engineering.* Electrical/Computer Science and Engineering Series. Van Noostrand Reinhold, New York, 1984.

[VW85] Vallacher, R.R. and Wegner, D.M., editors. *A theory of action identification.* Lawrence Erlbaum Associates, 1985.

[WBH87] Whiteside, J., Benett, J. and Holtblatt, K. Usability engineering: Our experience and evolution. In M. Helander, editor, *Handbook of human-computer interaction.* North-Holland Press, 1987.

[WF86] Winograd, T. and Flores, F. *Understanding Computers and Cognition.* Ablex, 1986.

[Whi27] Whitehead, A.N. *Symbolism, its meaning and effect.* Macmillan, New York, 1927.

[Wil84] Wilkins, D.E. Domain independent planning: Representation and plan generation. *Artificial Intelligence*, 1984.

[Woo86] Woods, D.D. Cognitive technologies: The design of joint human-machine cognitive systems. *The AI Magazine*, 6(4):86–92, 1986.

[WW82] Williges, B. and Williges, R. User consideration in computer based information systems. Technical Report csie-81-2, Naval Research Office, 1982.

[WW87] Whiteside, J. and Wixon, D. Discussion: Improving human computer interaction - a quest for cognitive science. In J. Carroll, editor, *Interfacing Thought.* MIT Press, 1987.

[Zis77] Zisman, M. *Representation, Specification and Automation of Office Procedures.* PhD thesis, Wharton School, University of Pennsylvania, 1977.

[Zlo75] Zloof, M. Query By Example. In *Proc. AFIPS NCC 44*, 1975.

[Zlo82] Zloof, M. Office by example: A business language that unifies data and word processing and electronic mail. *IBM Systems Journal*, 21, 1982.

[ZW86] Zdonik, S.B. and Wegner, P. Language and methodology for object-oriented database environments. In *Proceedings of the Nineteenth Annual Hawaii Internation Conference on System Sciences*, 1986.